SECOND INTERIM REPORT

on a

CODING SYSTEM DESIGN FOR ADVANCED SOLAR MISSIONS

Contract NAS2-3637

Submitted to:

Ames Research Center
National Aeronautics and Space Administration

May 12, 1967

Submitted by:

Codex Corporation
222 Arsenal Street
Watertown, Massachusetts 02172

Table of Contents

Table of Contents (Continued)

Introduction

This Second Interim Report covers work performed since the submission of the first Interim Report on Contract NAS2-3637. A summary of this work follows. The detailed studies appear in a series of appendices.

Summary

The first Interim Report summarized the results of simulations of OED schemes, by Codex, and of sequential decoding, by Ames. Because the sequential decoding scheme was 2-3 db superior to the best OED and 4-5 db superior to the present system, Ames determined that all further effort should be concentrated on sequential decoding. For systems reasons, a convolutional code of rate 1/2 and constraint length 25 was chosen.

We have rewritten our 7094 machine language sequential decoding simulator in more intelligible (macro) form, and in two versions, using the Gallager and the Fano algorithms. The capability of reverse decoding for error detection has been added, and a program to implement the error detection scheme proposed in Appendix C has been written. This program has been and is continuing to be used at Ames to pin down details of the decoding algorithm.

Ames first used this program to choose the best code of constraint length 25; the result was the length 24 code of Appendix D of the first Interim Report, with an additional tap in the 25th position. An encoder for this code, with the additional timing, modulating, and other functions described in the above-referenced appendix, has been constructed in breadboard form, checked out against a DTU simulator furnished by Ames, and delivered to Ames. This breadboard contains 92 integrated circuits of Texas Instruments' Series 51, and a small power supply. Worst case power consumption is 206 mw. At Ames' request the breadboard documentation package has been shipped separately with the breadboard and is therefore not included in this report.

Decoding may be accomplished by the SDS 910-920 computers presently at each of the antenna sites, by new small third-generation general-purpose computers, by special-purpose digital devices, or by a combination of the above. We have been active in all these approaches.

The SDS 910-920 program reported previously has been interfaced by Ames into a complete decoding system, which has been used for real-time and real-data simulations and demonstrations. We provide in this report complete documentation on the core sequential decoding program, in Appendix A. In Appendix B we present possible solutions to the important peripheral problems of input-output and timing, synchronization, and detection of decoder errors.

Appendix C evaluates some of the small general-purpose computers which appear to be likely candidates for the sequential decoding application. We find that a machine in the 1-microsecond, $30,000 class is capable of performing a single decoder computation in about 30 microseconds, as compared to about 200 us for the SDS 910-920; these figures ignore all overhead and special features of the Pioneer application.

Finally, in Appendix D, we present the results of a detailed study on the capabilities and cost of a special-purpose machine for this application. We find that one can achieve decoder computation times of about 1 microsecond with readily available components, principally integrated circuits and a small core memory, at a total hardware materials cost under $10,000. Speeds down to 100 nanoseconds appear possible.

Future Work

This completes Codex' work under this contract on the development of a coding system for Pioneer missions of the near future. It has been a fascinating, exciting, and satisfying business, and while we are elated to see it advancing so rapidly toward actual use, we are a bit regretful that our part of it is over. The remainder of our work under this contract will be devoted to the study of concatenation with sequential decoding, in the hope that for the advanced missions of the future we can approach even more closely the ultimate limits of efficiency on communication from deep space.

APPENDIX A

SEQUENTIAL DECODING PROGRAM FOR THE SDS 910-920

In the previous Interim Report, Appendix C, we reported the writing of two sequential decoding programs for the SDS 910 or 920, a "fast" program and a "short" program. The "short" program has now been implemented, with all needed interfacing, by L. Hofman of Ames and is intended for actual use in the Pioneer program. We therefore provide here a detailed description of this program, including a conceptual description, flow charts, and a commented listing.

Conceptual Description of the Program

Sequential decoding is a programmed tree search. In Figure 1 we illustrate a small part of the tree for a rate 1/2 code. Each branch is characterized by a single information bit, or by two transmitted bits, the first being the information bit, and the second a parity bit P which is the mod 2 sum of the information bit with a subset of the previous 24 bits in the tree.

The basic unit of the algorithm is a move, or trial; in one move the decoder looks at the information pertinent to one information bit, called the current bit, decides whether to make a tentative decision on the current bit and proceed to the next bit or to retreat to the previous bit, and updates the bookkeeping data which reflect its decision. We see that in general the decision is trivalued: back, forward on 0, or forward on 1. However, when certain bits are known, or bear a known relationship to previous bits, as with word parity checks, only one of the two forward moves may be permissible.

The basis for these decisions is the decoding metric. The metric for any particular branch is a function of the two transmitted bits describing that branch and the six bits describing what was received at that time. We assume that the actual values of the metric for these 256 possible combinations have been precomputed and stored in four 64-word tables. We find it convenient to require that all metrics be even integers (which involves no loss in precision).

As the decoder searches through the tree, it keeps track of a quantity which we call the adjusted metric, which may be thought of as the sum of all branch metrics up to the current branch, minus some integer times a constant, the threshold spacing DELTA, which must be an even integer but is otherwise an arbitrary parameter of the algorithm. The decision and adjustment of the metric then follow the following rules, which are modifications of those due to Gallager. In this table M_o is the metric of the branch with a 0 information bit, M_1, the one information bit, and METRIC the accumulated adjusted metric at the start of the move. [X] is the (greatest negative integer) least positive integer of the form X-i*DELTA. A previous move of back (0) indicates that the previous move was back to a branch specified by an information bit of 0, and so forth.

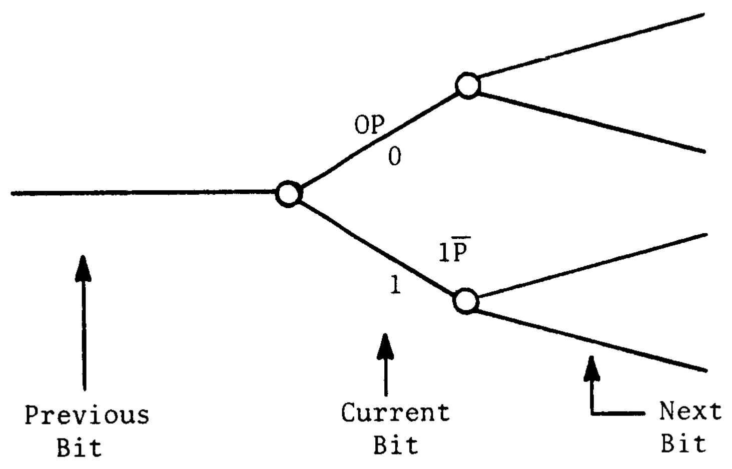


Figure A-1

Name	Previous Move	Metric Conditions		Decision	Metric Adjustment
PF0	forward	METRIC+$M_0 > 0$,	METRIC < DELTA	forward on 0	METRIC=[METRIC+M_0]
SF0	forward	METRIC+$M_0 > 0$,	METRIC > DELTA	forward on 0	METRIC= METRIC+M_0
PF1	forward	METRIC+$M_0 < 0$, METRIC+$M_1 > 0$,	METRIC < DELTA	forward on 1	METRIC=[METRIC+M_1]
BF1	forward	METRIC+$M_0 < 0$, METRIC+$M_1 > 0$,	METRIC > DELTA	forward on 1	METRIC= METRIC+M_1
FB	forward	METRIC+$M_0 < 0$, METRIC+$M_1 < 0$		back	METRIC= METRIC
BF0	back (0)	METRIC-$M_0 < 0$		forward on 0	METRIC= METRIC+DELTA
BF1	back (1)	METRIC-$M_1 < 0$		forward on 1	METRIC= METRIC+DELTA
BLPF	back (0)	DELTA > METRIC-$M_0 > 0$,	METRIC-$M_0+M_1 > 0$	forward on 1	METRIC=[METRIC-M_0+M_1]
BLSF	back (0)	METRIC-M_0 > DELTA,	METRIC-$M_0+M_1 > 0$	forward on 1	METRIC= METRIC-M_0+M_1
BB0	back (0)	METRIC-$M_0 > 0$,	METRIC-$M_0+M_1 < 0$	back	METRIC= METRIC-M_0
BB1	back (1)	METRIC-$M_1 > 0$		back	METRIC= METRIC-M_1

The initial value for METRIC is set at (-1) 1; with M_0 and M_1 always even, METRIC will always be odd, and thus comparisons with 0 and DELTA never result in equality. By examination of the rules we see that METRIC is also always (negative) positive. If the algorithm ever attempts to move back from the first bit in the tree, we force a BF move; if it ever moves forward beyond the last bit in the tree, decoding is complete.

All that remains to specify the basic algorithm is to give a method of telling the algorithm where it is in the tree. On a forward move, the decoder must know the parity bit corresponding to a 0 information bit (that corresponding to 1 will be the complement); on a backward move, it must know the information and parity bits which describe the previous branch. The former problem is handled by storing a word called PARITY, which tells what the next constraint length of parity bits will be if all information bits are zeroes. We find it convenient to let the left-most (sign) bit of the parity word correspond to the current bit, after a forward move, and to the next bit, after a backward move; this saves a little time. PARITY is initially set to all zeroes. Whenever a 1 is tentatively decided on, a 24-bit sequence TAP 1, representing the impulse response of a single 1, is added to PARITY bit-by-bit and mod 2; should such a decision be revoked, TAP 1 is added again, and thus cancels. Also, PARITY must be left- and right-shifted as the algorithm moves backward and forward. The result is that we implement what in Massey's terms would be a Type II encoder.

To tell what the previous branch was, we use a 224-word table called EST(N). On a forward move from bit N, we store in EST(N) one of four addresses, corresponding to the four possible combinations of information and parity. Then if we should ever move back to bit N, we use an indirect address to transfer into the appropriate one of four subprograms, thus saving all testing. Furthermore, at the end of decoding the frame, we can read the decoded information from the EST(N) table; this readout is facilitated if all addresses corresponding to a 0 information bit are positive, and those to 1, negative.

The algorithm described so far is a complete decoding algorithm for a rate 1/2 code of constraint length 25 with a resynchronizing frame of 224 bits. However, the rigidities of the Pioneer format give us additional information which it is to our advantage to use in decoding. These are the several fixed words which occur in each frame, and the over-all parity check which is added to most words. We have classified all bits in the frame into one of the following categories:

1. A fixed bit equal to 1;
2. A fixed bit equal to 0;
3. A bit which is a parity check on preceding bits;
4. A bit which enters into a parity check;
5. The first bit in a word, assumed entering into a parity check;
6. All other bits.

Thus a fixed word is described by a mixture of Types 1 and 2; a word with over-all parity check by 5444443; and a word with 1-3-5-7 parity by 5646463. The description of the entire frame then fills a 224-word table MOVEF(N), which we again translate into addresses corresponding to six subprograms; a forward move is then begun by an indirect address transfer from this table, just like a backward move. In addition, the backward move addresses must be increased to describe a greater variety of possible backward moves.

The only change required by a fixed bit of 0, say, is that, of the forward moves, PF1 and SF1 are no longer permitted, and the condition for an FB move becomes simply METRIC + $M_o < 0$; similarly BLPF and BLSF are excluded as backward moves, and the condition for BB0 is METRIC - $M_o > 0$. For the bits involved in parity checks, we introduce an integer variable P7; P7 is set to 0 or 1 on the first bit in each word, depending on whether an even or odd parity check is desired; it is then incremented by 1 every time a bit is decided to be a 1, or changed from a 1 to 0. Then, when the parity check bit arrives, P7 is tested for odd or evenness, and depending on the outcome, the bit is treated as a fixed 1 or 0.

Flow Charts and Listing

There follow detailed flow charts for the actual program, and a commented listing. The flow charts are preceded by a glossary of the different entries which the program may reach on forward or backward moves. Then, in order to save needless duplication, we have written out as macro subprograms the three general types of updating routine needed before a forward move. The detailed flow charts for each entry then follow, and finally the listing.

TABLE I

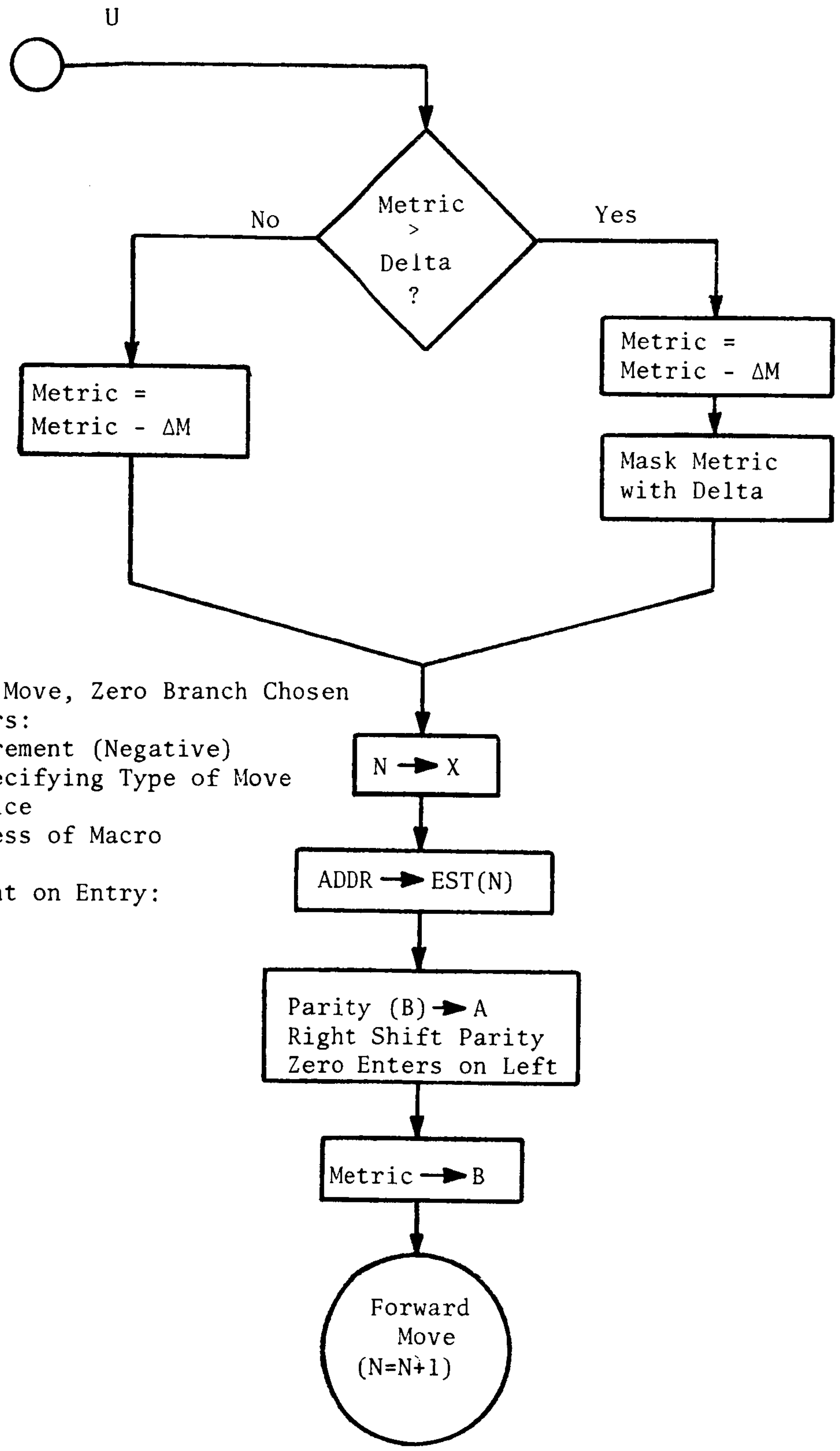
FORWARD MOVE ENTRIES

E1	(AA1)	first bit in word
EF	(AA2)	ordinary bit in parity check (third, fifth bits)
EF7	(AA3)	last bit in word
EFX0	(AA4)	fixed bit = 0
EFX1	(AA5)	fixed bit = 1
EF	(AA6)	ordinary bit not in parity check (second, fourth, sixth)
OUT	(AA7)	bit after last bit in frame

TABLE II

BACKWARD MOVE ENTRIES

B00	(A1)	ordinary bit, 00 chosen
B01	(A2)	ordinary bit, 01 chosen
B10	(A3)	ordinary bit, 10 chosen
B11	(A4)	ordinary bit, 11 chosen
BX00	(A5)	fixed bit, 00 chosen
BX01	(A6)	fixed bit, 01 chosen
B700	(A7)	seventh bit, 00 chosen
B701	(A8)	seventh bit, 01 chosen
B710	(A9)	seventh bit, 10 chosen
B711	(A10)	seventh bit, 11 chosen
BINIT	(A11)	bit before the first bit in the frame

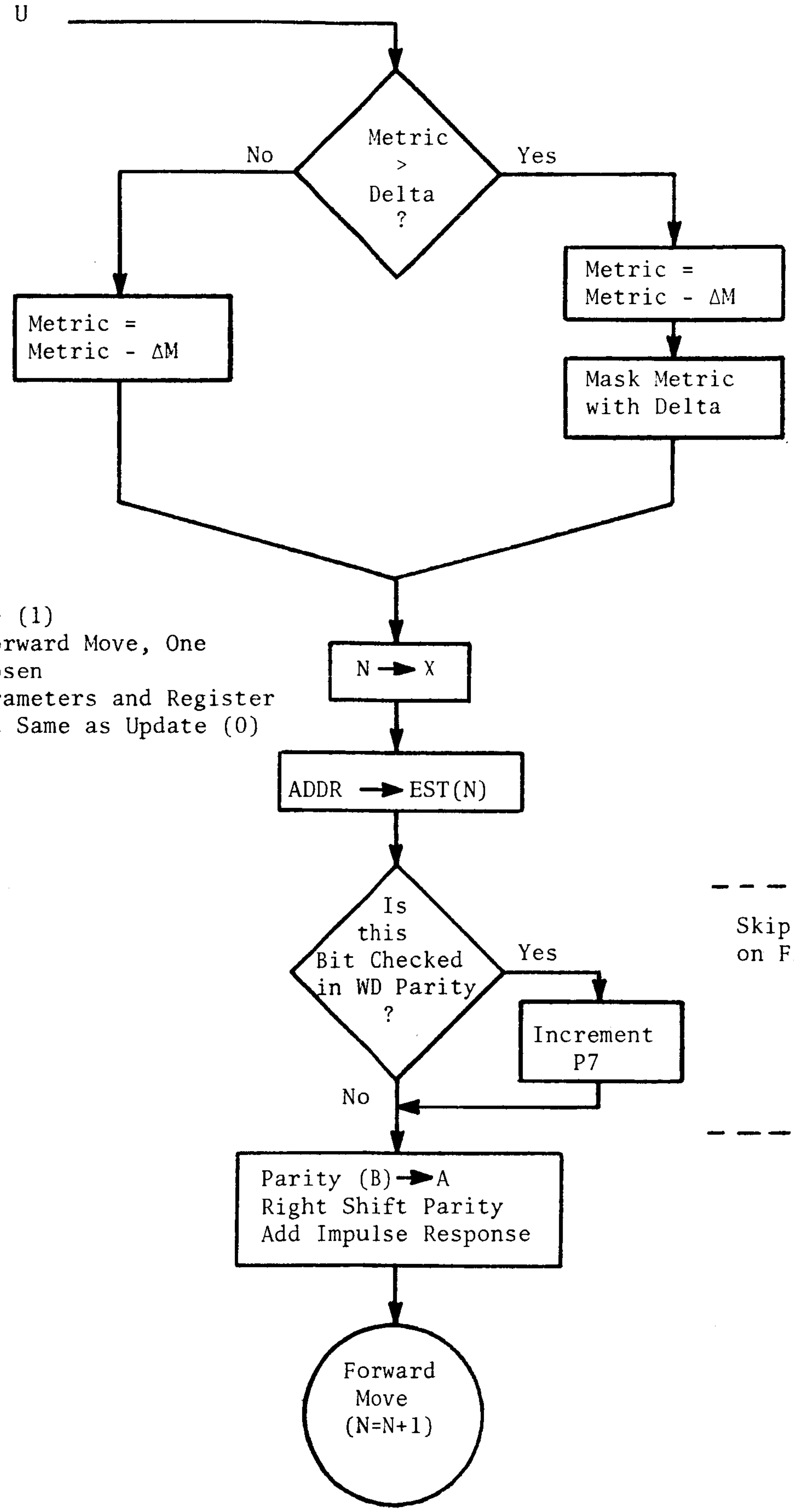


Macro Update (0)
Update on Forward Move, Zero Branch Chosen
Variable Parameters:

ΔM:	Metric Increment (Negative)
ADDR:	Address Specifying Type of Move and Choice
U:	Entry Address of Macro

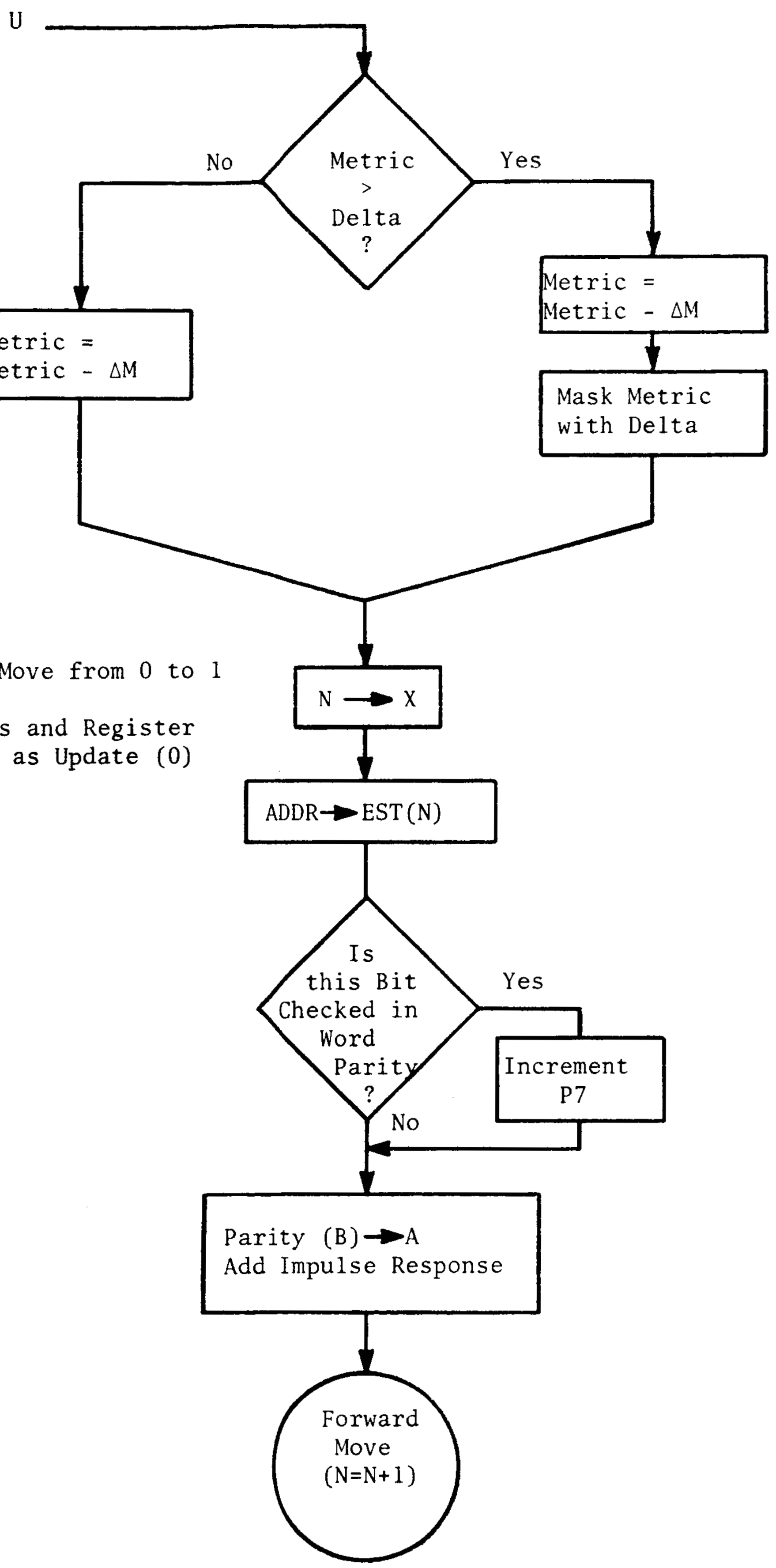
Register Assignment on Entry:

A:	Old Metric
B:	Old Parity
X:	Data (N)



Macro Update (1)
Update on Forward Move, One Branch Chosen
Variable Parameters and Register Assignment Same as Update (0)

Skip this Section on Fixed Bits

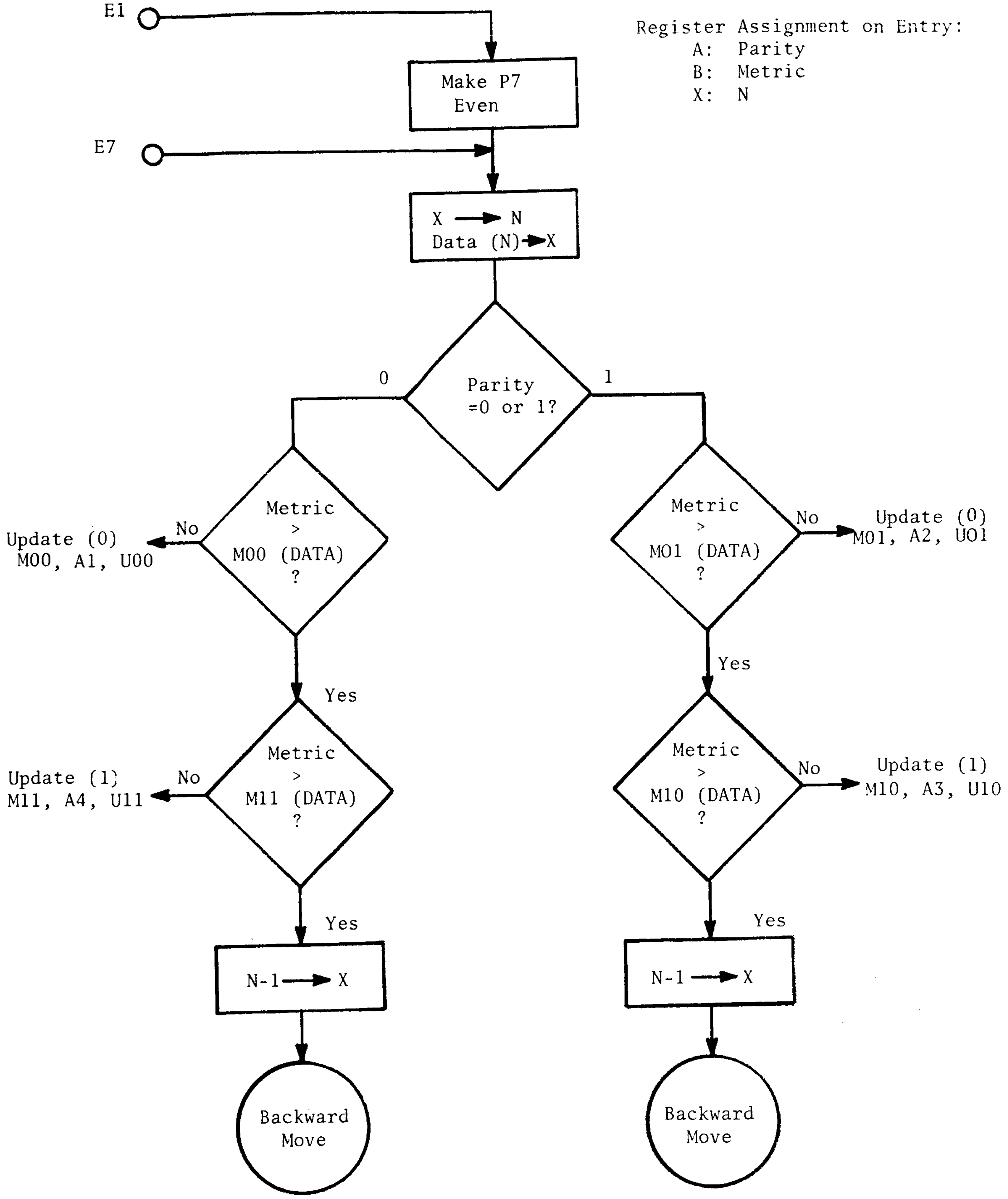


Macro Update (L)
Update on Lateral Move from 0 to 1 Branch
Variable Parameters and Register Assignment Same as Update (0)

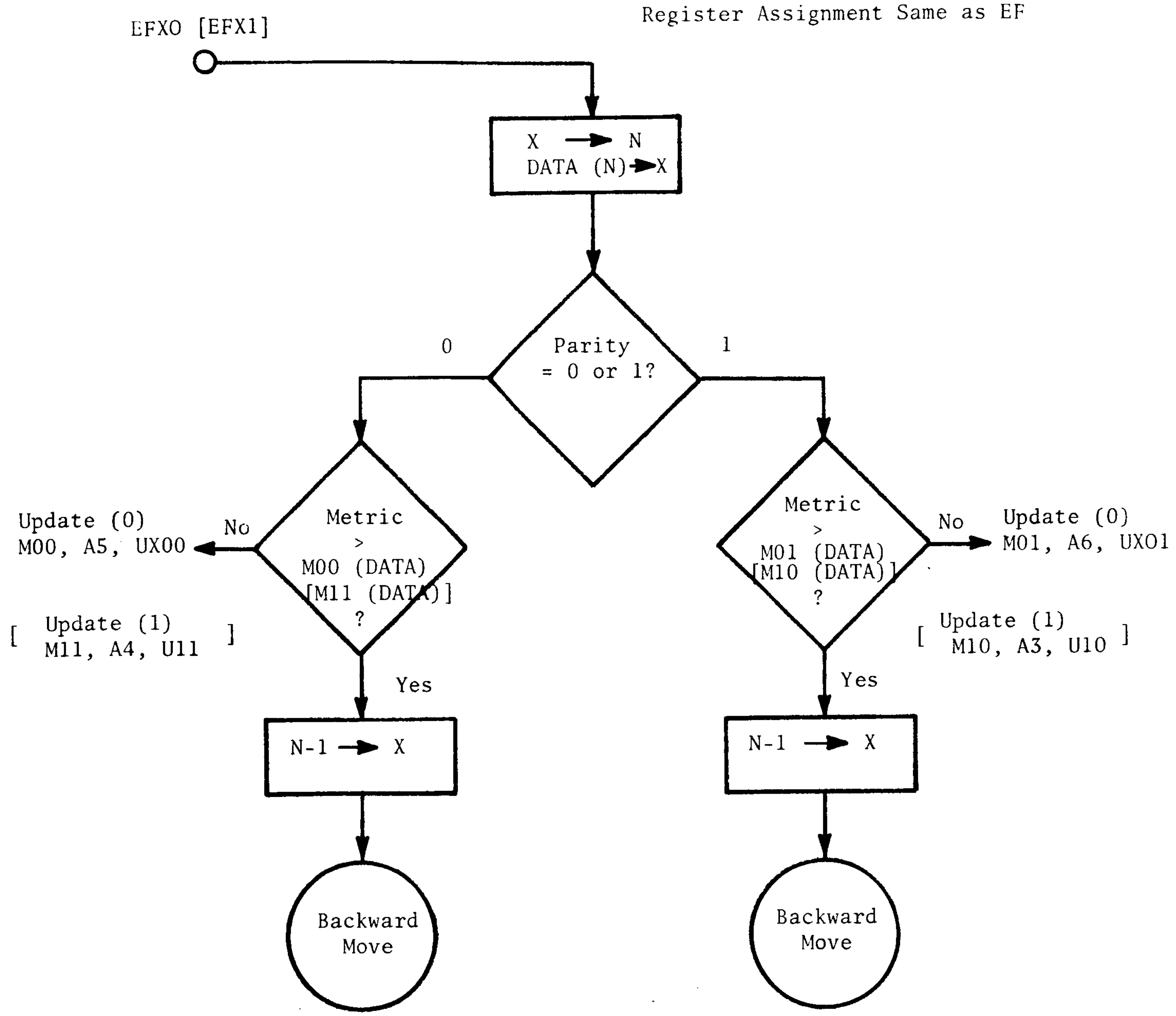
Forward Moves: Entries E1 and EF

Register Assignment on Entry:

A:	Parity
B:	Metric
X:	N



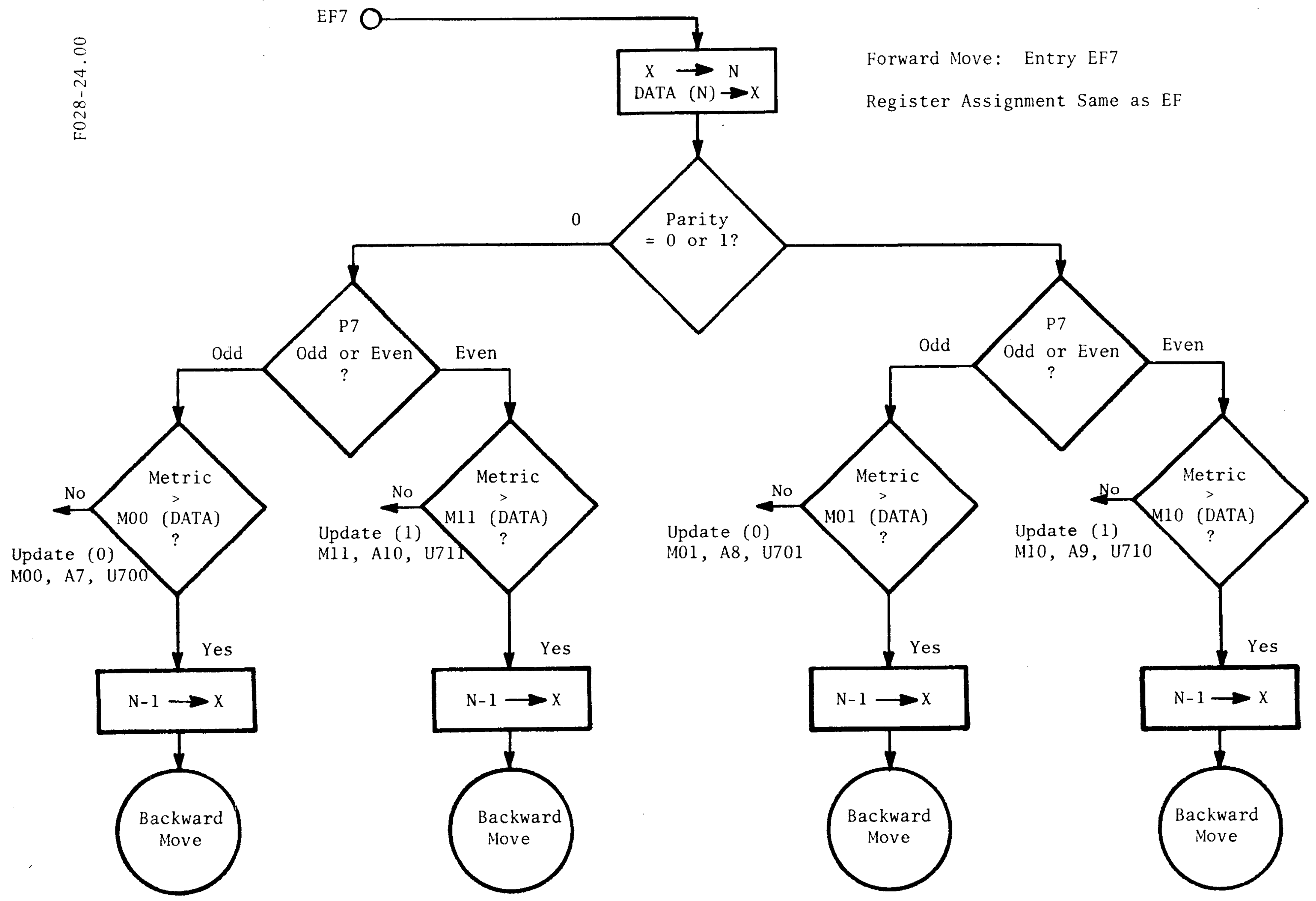
Forward Moves: Entries EFXO [EFX1]
Register Assignment Same as EF



F028-24.00

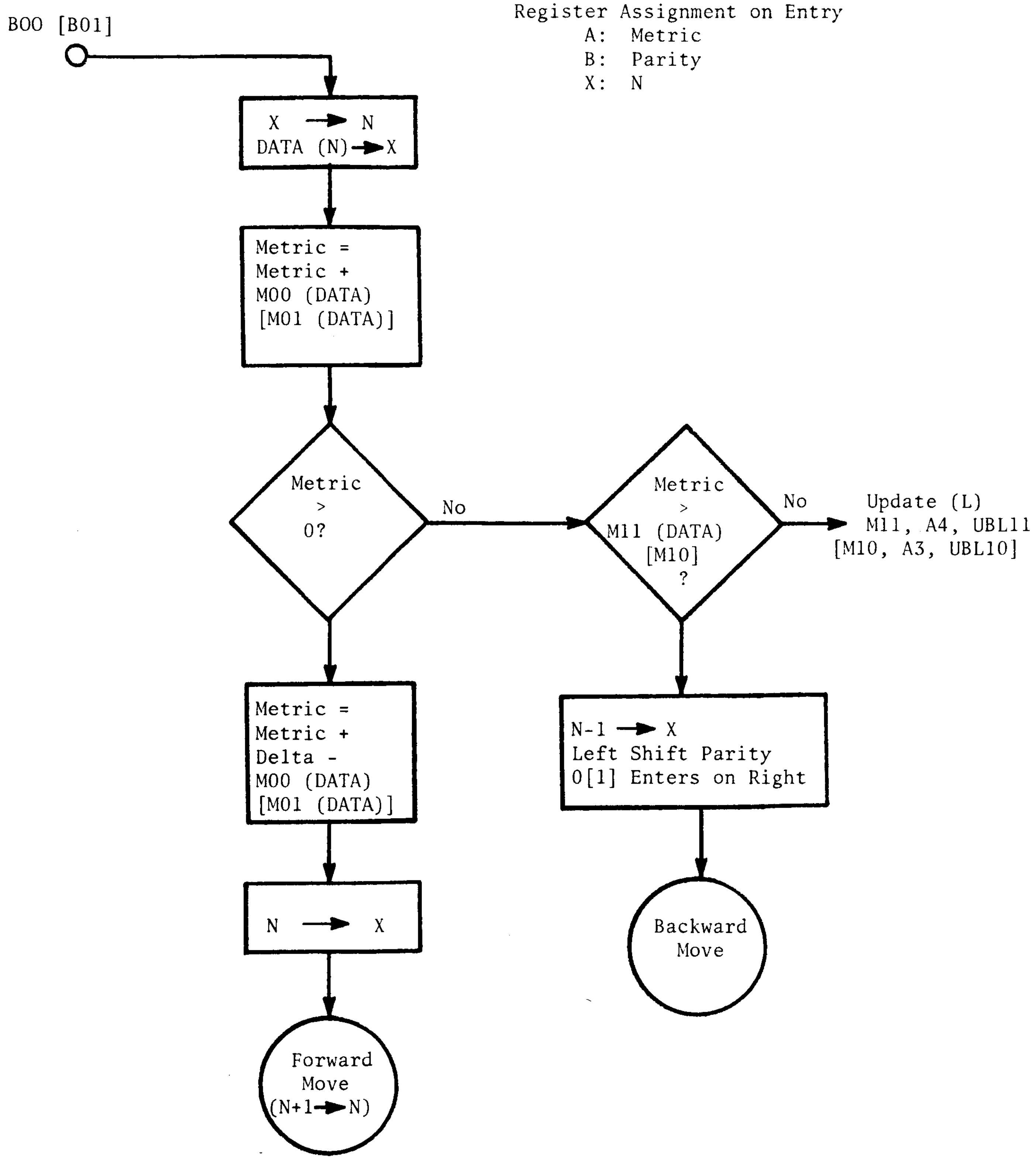
Forward Move: Entry EF7

Register Assignment Same as EF



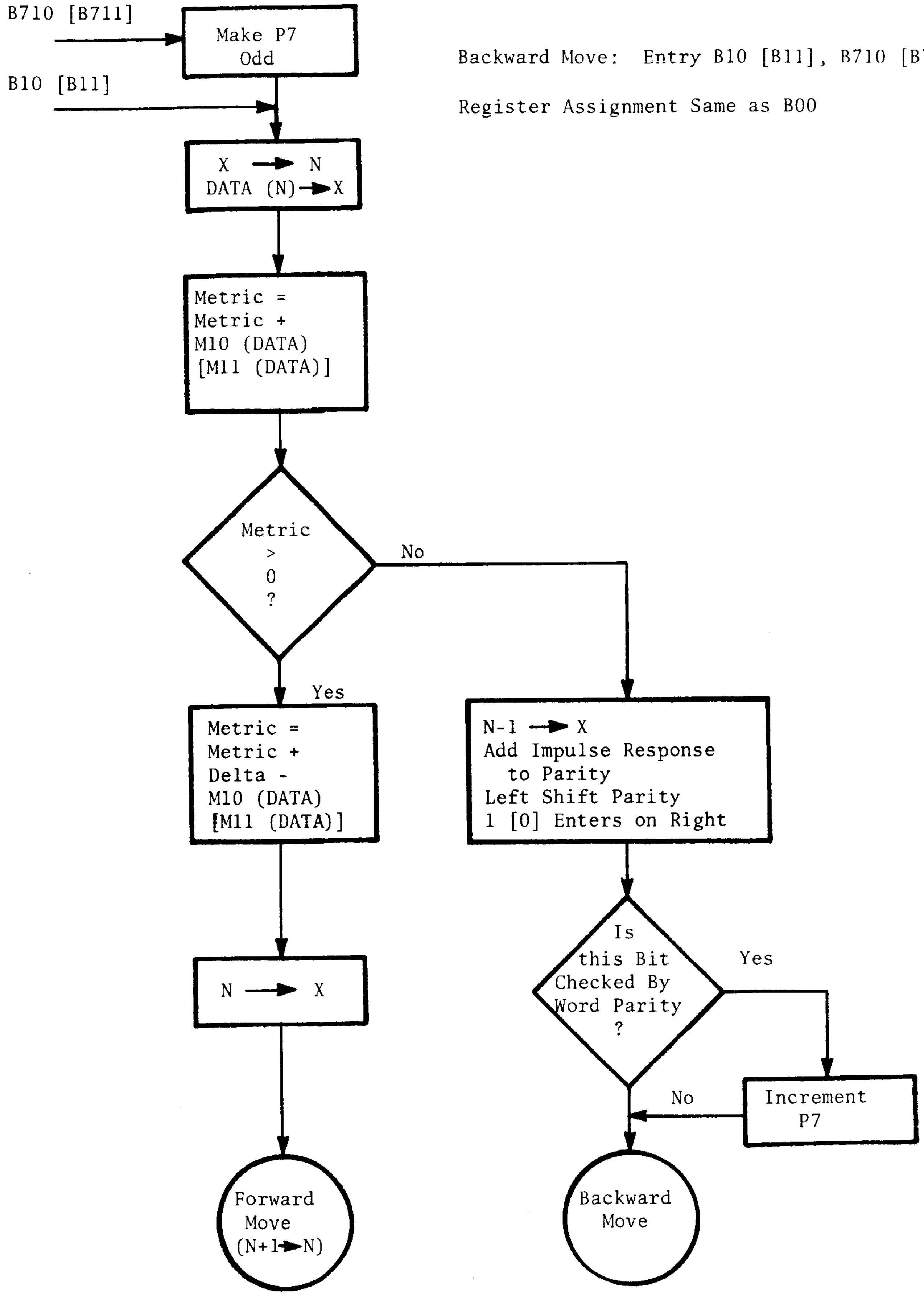
Backward Move: Entry B00 [B01]

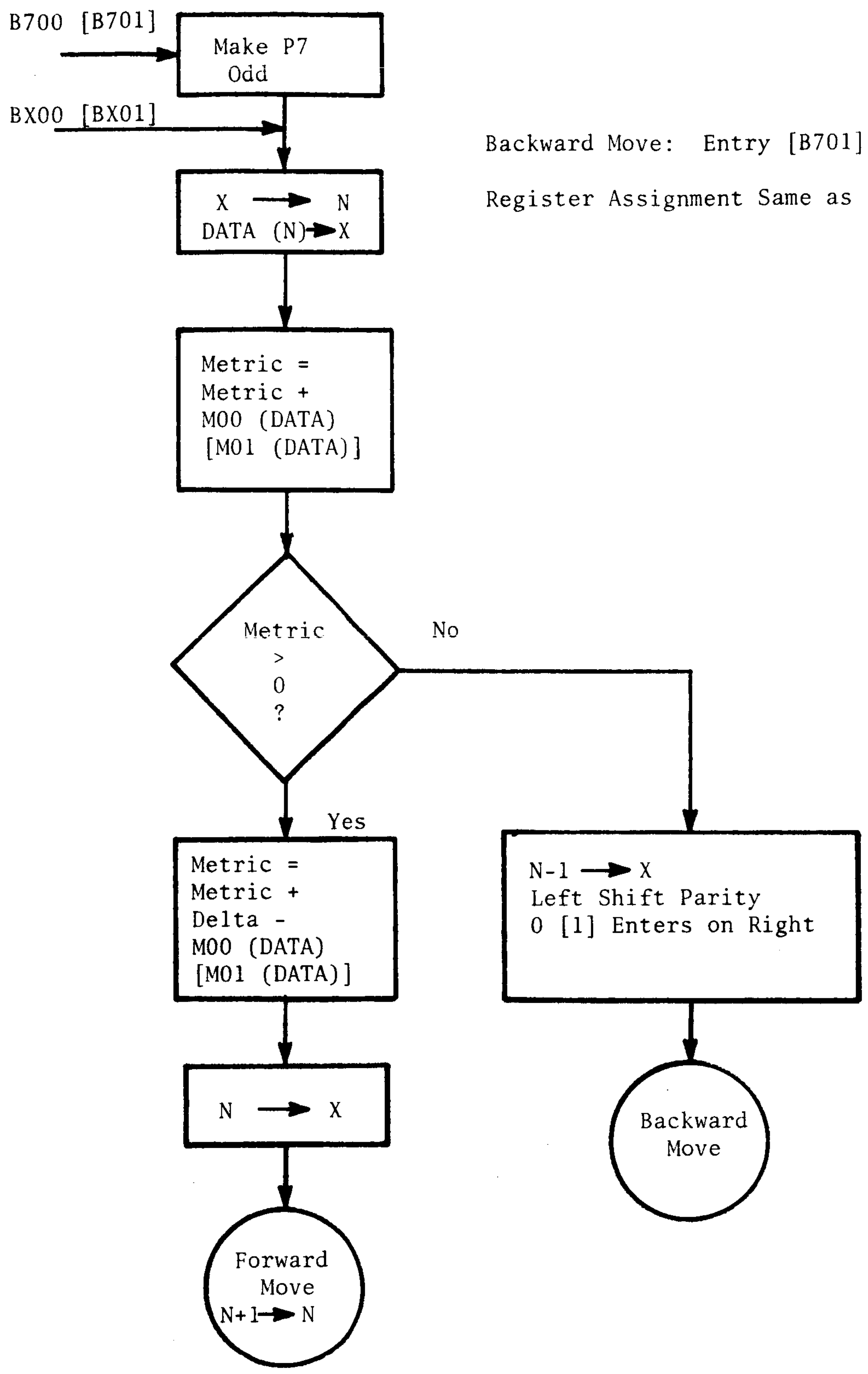
Register Assignment on Entry
A: Metric
B: Parity
X: N


B00 [B01]
X → N
DATA (N) → X
Metric = Metric + M00 (DATA) [M01 (DATA)]
Metric > 0?
No
Metric > M11 (DATA) [M10] ?
No
Update (L) M11, A4, UBL11 [M10, A3, UBL10]
Metric = Metric + Delta - M00 (DATA) [M01 (DATA)]
N → X
Forward Move (N+1 → N)
N-1 → X
Left Shift Parity
0[1] Enters on Right
Backward Move

Backward Move: Entry B10 [B11], B710 [B711]

Register Assignment Same as B00





Backward Move: Entry [B701], BX00 [BX01]

Register Assignment Same as B00

SDS 910-920 Sequential Decoder Program Listing

Initialization

```
              LDX        =-10         \
              LDA        A11, 2        |
              ETR        SGNMSK        |
              STA        A11, 2        |
              BRX        $-3           |
              LDA        A3            |
              MRG        SGNBIT        |
              STA        A3            > A3, A4, A9, A10 SET NEGATIVE
              LDA        A4            | REST OF A1-A10 POSITIVE
              MRG        SGNBIT        |
              STA        A4            |
              LDA        A9            |
              MRG        SGNBIT        |
              STA        A9            |
              LDA        A10           |
              MRG        SGNBIT        |
              STA        A10          /
              LDX        =-3          \
              LDA        AA3+1, 2      |
              ETR        SGNMSK        > AA1, AA2, AA3 POSITIVE
              STA        AA3+1, 2      |
              BRX        $-3          /
              LDA        AA6          \
              MRG        SGNBIT        > AA6 SET NEGATIVE
              STA        AA6          /
              LDX        =-224        \
     ITOP     LDA        AA1           |
              STA        MOVEF-2, 2    |
              BRX        $+1           |
              LDA        AA6           |
              STA        MOVEF-2, 2    |
              BRX        $+1           |
              LDA        AA2           |
              STA        MOVEF-2, 2    |
              BRX        $+1           |
              LDA        AA6           > SET UP MOVEF TABLE, PATTERN IS
              STA        MOVEF-2, 2    | AA1, AA6, AA2, AA6, AA2, AA6, AA3
              BRX        $+1           |
              LDA        AA2           |
              STA        MOVEF-2, 2    |
              BRX        $+1           |
              LDA        AA6           |
              STA        MOVEF-2, 2    |
              BRX        $+1           |
              LDA        AA3           |
              STA        MOVEF-2, 2    |
              BRX        ITOP         /
```

```
        LDA     AA4              \
        LDX     =-7              |
        STA     MOVEF-121, 2     |
        BRX     $-1              |
        LDX     =-11             |
        STA     MOVEF-2, 2       |
        BRX     $-1              |
        LDA     AA5              > OVERWRITE FIXED WORDS IN MOVEF
        STA     MOVEF-125        |
        STA     MOVEF-124        |
        STA     MOVEF-122        |
        STA     MOVEF-16         |
        STA     MOVEF-15         |
        STA     MOVEF-14         |
        STA     MOVEF-11         /
        LDA     AA7              INSERT OUT INSTRUCTION AT END OF
        STA     MOVEF-2          MOVEF TABLE
        LDA     A11              INSERT BINIT AT BEGINNING OF EST
        STA     EST-226          TABLE
```

Core of the Program

```
BEGIN   CLR                      BEGIN FRAME DECODING--PARITY=0 TO STA
        LDX     =-226            N=-226, COUNTS UP TO -1
        SUB     LOBIT            METRIC=-1, METRIC ALWAYS - AND ODD
        XAB
        BRX     *MOVEF, 2
E1      STB     P7               FIRST BIT ENTRY, FORWARD MOVE
        MIN     P7               P7 SET EVEN.  FOR EVEN PARITY NOP INS
EF      STX     N                ORDINARY BIT ENTRY, FORWARD MOVE
*       PARITY IN A, METRIC IN B, N IN X.
        LDX     DATA, 2
        SKA     LOBIT            DETERMINE CURRENT PARITY BIT
        BRU     EF1
        XAB
        SKG     M00, 2           IS METRIC-M00(DATA).LT.0...
        BRU     U00              YES, DECIDE TO MOVE FORWARD, UPDATE
        SKG     M11, 2           NO, TRY OTHER BRANCH
        BRU     U11
        LDX     N                BOTH BRANCHES BAD
        EAX     -1, 2            DECREMENT N
        BRU     *EST, 2          GO TO THE APPROPRIATE BACKWARD MOVE
EF1     XAB
        SKG     M01, 2
        BRU     U01
        SKG     M10, 2
        BRU     U10
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EFX0    STX     N           FIXED 0 BIT ENTRY, SAME AS EF EXCEPT
        LDX     DATA, 2       INHIBITS SECOND BRANCH
        SKA     LOBIT
        BRU     EFX01
        XAB
        SKG     M00, 2
        BRU     UX00
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EFX01   XAB
        SKG     M01, 2
        BRU     UX01
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EFX1    STX     N           FIXED 1 BIT ENTRY
        LDX     DATA, 2
        SKA     LOBIT
        BRU     EFX10
        XAB
        SKG     M11, 2
        BRU     U11
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EFX10   XAB
        SKG     M10, 2
        BRU     U10
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EF7     STX     N           SEVENTH BIT ENTRY
        LDX     DATA, 2
        STB     METRIC
        SKA     LOBIT       TEST CURRENT PARITY BIT
        BRU     EF71
        XAB
        LDA     P7
        SKA     LOBIT       TEST WHETHER P7 ODD OR EVEN
        BRU     EF700       P7 ODD, FIX SEVENTH BIT 0
        LDA     METRIC      P7 EVEN, FIX SEVENTH BIT 1
        SKG     M11, 2
        BRU     U711
        LDX     N
        EAX     -1, 2
        BRU     *EST, 2
EF700   LDA     METRIC
```

```
        SKG      M00, 2
        BRU      U700
        LDX      N
        EAX      -1, 2
        BRU      *EST, 2
EF71    XAB
        LDA      P7
        SKA      LOBIT
        BRU      EF701
        LDA      METRIC
        SKG      M10, 2
        BRU      U710
        LDX      N
        EAX      -1, 2
        BRU      *EST, 2
EF701   LDA      METRIC
        SKG      M01, 2
        BRU      U701
        LDX      N
        EAX      -1, 2
        BRU      *EST, 2
U00     SKG      DELTA          UPDATE PARAM, 00 CHOSEN, NORMAL BIT
        BRU      SF00           IF METRIC.LT.DELTA INHIBIT TH RAISE
        SUB      M00, 2         COMPUTE NEW METRIC
        MRG      MASK           RAISE TH, AS A RESULT DELTA.LT.M.LT.0
R00     STA      METRIC
        LDA      A1             LOAD ADDRESS WITH U00, STORE IN EST
F0      LDX      N
        STA      EST, 2
        BAC                     RIGHT SHIFT PARITY 1 INSERT 0 IN LEFT
        RCY      1
        LDB      METRIC
        BRX      *MOVEF, 2      GO TO FORWARD ENTRY
SF00    SUB      M00, 2         BYPASSES MRG MASK
        BRU      R00
U01     SKG      DELTA          UPDATE, 01 CHOICE, ORDINARY BIT
        BRU      SF01
        SUB      M01, 2
        MRG      MASK
R01     STA      METRIC
        LDA      A2
        BRU      F0
SF01    SUB      M01, 2
        BRU      R01
U10     SKG      DELTA          UPDATE, 10 CHOICE, ORDINARY BIT
        BRU      SF10
        SUB      M10, 2
        MRG      MASK
R10     STA      METRIC
        LDA      A3
F1      LDX      N
```

```
        STA     EST, 2
        SKN     MOVEF-1, 2      CHANGE PARITY OF P7 IF CURRENT BIT
        MIN     P7                IN PARITY CHECK
F1A     BAC
        RCY     1
        EOR     TAP3             ADD IMPULSE RESPONSE TO PARITY
        LDB     METRIC
        BRX     *MOVEF, 2
SF10    SUB     M10, 2
        BRU     R10
U11     SKG     DELTA            UPDATE, 11 CHOICE, ORDINARY LIST
        BRU     SF11
        SUB     M11, 2
        MRG     MASK
R11     STA     METRIC
        LDA     A4
        BRU     F1
SF11    SUB     M11, 2
        BRU     R11
UX00    SKG     DELTA            UPDATE, 00 CHOICE, FIXED BIT
        BRU     SFX00
        SUB     M00, 2
        MRG     MASK
RX00    STA     METRIC
        LDA     A5
        BRU     F0
SFX00   SUB     M00, 2
        BRU     RX00
UX01    SKG     DELTA            UPDATE, 01 CHOICE, FIXED BIT
        BRU     SFX01
        SUB     M01, 2
        MRG     MASK
RX01    STA     METRIC
        LDA     A6
        BRU     F0
SFX01   SUB     M01, 2
        BRU     RX01
U700    SKG     DELTA            UPDATE, 00 CHOICE, SEVENTH BIT
        BRU     SF700
        SUB     M00, 2
        MRG     MASK
R700    STA     METRIC
        LDA     A7
        BRU     F0
SF700   SUB     M00, 2
        BRU     R700
U701    SKG     DELTA            UPDATE, 01 CHOICE, SEVENTH BIT
        BRU     SF701
        SUB     M01, 2
        MRG     MASK
R701    STA     METRIC
```

```
         LDA      A8
         BRU      F0
SF701    SUB      M01, 2
         BRU      R701
U710     SKG      DELTA           UPDATE, 10 CHOICE, SEVENTH BIT
         BRU      SF710
         SUB      M10, 2
         MRG      MASK
R710     STA      METRIC
         LDA      A9
         LDX      N
         STA      EST, 2
         BRU      F1A
SF710    SUB      M10, 2
         BRU      R710
U711     SKG      DELTA           UPDATE, 11 CHOICE, SEVENTH BIT
         BRU      SF711
         SUB      M11, 2
         MRG      MASK
R711     STA      METRIC
         LDA      A10
         LDX      N
         STA      EST, 2
         BRU      F1A
SF711    SUB      M11, 2
         BRU      R711
B00      STX      N               BACK ENTRY, 00 BRANCH, NORMAL BIT
         LDX      DATA, 2         METRIC IN A, PARITY IN B, N IN X
         ADD      M00, 2          RESTORE EARLIER METRIC
         SKA      SGNBIT          IS EARLIER METRIC .GT.0
         BRU      L00             NO, TRY A LATERAL MOVE
FB00     SUB      M00, 2          YES, TURN AROUND, RESTORE LATER METRIC
FBA      LDX      N               AND LOWER THRESHOLD BY DELTA
BINIT    ADD      DELTA
         XAB
         BRX      *MOVEF, 2
L00      SKG      M11, 2          LATERAL TRY, 00 BRANCH, IS M-M11.LT.0
         BRU      UBL11           YES, GO TO UPDATE
UBX00    STA      METRIC          NO, GO BACK AGAIN
         LDX      N
         NOD      1               LEFT SHIFT PARITY, DECREMENT N,
         LDA      METRIC          0 ENTERS ON RIGHT
         BRU      *EST, 2
B01      STX      N               BACK ENTRY, 01 BRANCH, NORMAL BIT
         LDX      DATA, 2
         ADD      M01, 2
         SKA      SGNBIT
         BRU      L01
         SUB      M01, 2
         BRU      FBA
L01      SKG      M10, 2
```

```
        BRU     UBL10
UBX01   STA     METRIC
        LDX     N
        LCY     1               LEFT SHIFT PARITY, 1 ENTERS ON RIGHT
        LDA     METRIC
        EAX     -1, 2           DECREMENT N
        BRU     *EST, 2
UBL11   SKG     DELTA           UPDATE, LATERAL MOVE TO 11 BRANCH
        BRU     SFBL11
        SUB     M11, 2
        MRG     MASK
RBL11   STA     METRIC
        LDA     A4
E11     LDX     N
        STA     EST, 2
        SKN     MOVEF-1, 2
        MIN     P7
        XAB
        EOR     TAP3            SAME AS U11 BUT DOES NOT SHIFT PARITY
        LDB     METRIC
        BRX     *MOVEF, 2
SFBL11  SUB     M11, 2
        BRU     RBL11
UBL10   SKG     DELTA           UPDATE, LATERAL MOVE TO 10 BRANCH
        BRU     SFBL10
        SUB     M10, 2
        MRG     MASK
RBL10   STA     METRIC
        LDA     A3
        BRU     E11
SFBL10  SUB     M10, 2
        BRU     RBL10
B710    STA     P7              BACK ENTRY, 10 BR, 7TH BIT, SET P7 ODD
B10     STX     N               BACK ENTRY, 10 BRANCH, NORMAL BIT
        LDX     DATA, 2
        ADD     M10, 2
        SKA     SGNBIT          EARLIER METRIC .GT.0
        BRU     UB10            YES, GO BACK
        SUB     M10, 2          NO, GO FORWARD AND LOWER THRESHOLD
        BRU     FBA
B711    STA     P7              BACK ENTRY, 11 BR, 7TH BIT, SET P7 ODD
B11     STX     N               BACK ENTRY, NORMAL BR, NORMAL BIT
        LDX     DATA, 2
        ADD     M11, 2
        SKA     SGNBIT
        BRU     UB11
        SUB     M11, 2
        BRU     FBA
UB10    STA     METRIC          GO BACK, UPDATE PARAMETERS
        XAB
        EOR     TAP3
        LSH     1             LEFT SHIFT PARITY, 1 ENTERS ON RIGHT
        XAB
        LDX     N
        SKN     MOVEF-1, 2
        MIN     P7
        LDA     METRIC
        EAX     -1, 2
        BRU     *EST, 2
UB11    STA     METRIC        GO BACK, UPDATE PARAMETERS
        BAC
        EOR     TAP3
        XAB
        LDX     N
        SKN     MOVEF-1, 2
        MIN     P7
        NOD     1             LEFT SHIFT PARITY, DECR N, 0 ON RIGHT
        LDA     METRIC        0 IN A PERMITS NORMALIZE TO BE COMP.
        BRU     *EST, 2
B700    STA     P7            BACK ENTRY, 00 BR, 7TH BIT, SET P7 ODD
BX00    STX     N             BACK ENTRY, 00BR, FIXED BIT
        LDX     DATA, 2
        ADD     M00, 2
        SKA     SGNBIT
        BRU     UBX00
        BRU     FB00
B701    STA     P7            BACK ENTRY, 01 BR, 7TH BIT, SET P7 ODD
BX01    STX     N             BACK ENTRY, 01 BR, FIXED BIT
        LDX     DATA, 2
        ADD     M01, 2
        SKA     SGNBIT
        BRU     UBX01
        SUB     M01, 2
        BRU     FBA
OUT     NOP                   DECODING COMPLETED
```

Constants

```
INFO    RES     224
B0      RES     226
DATA    EQU     B0+225
B1      RES     225
MOVEF   EQU     B1+226
B2      RES     225
EST     EQU     B2+226
TAP3    OCT     27072325      TAPS, R TO L, OMIT FIRST BIT
DELTA   OCT     77776000      MUST BE A POWER OF 2
MASK    EQU     DELTA
```

```
SGNBIT   OCT        40000000
LOBIT    OCT        1
SGNMSK   OCT        37777777
A1       BRU        B00
A2       BRU        B01
A3       BRU        B10
A4       BRU        B11
A5       BRU        BX00
A6       BRU        BX01
A7       BRU        B700
A8       BRU        B701
A9       BRU        B710
A10      BRU        B711
A11      BRU        BINIT
P7       RES        1
N        RES        1
METRIC   RES        1
M00      RES        64
M01      RES        64
M10      RES        64
M11      RES        66
AA1      BRU        E1
AA2      BRU        EF
AA3      BRU        EF7
AA4      BRU        EFX0
AA5      BRU        EFX1
AA6      BRU        EF
AA7      BRU        OUT
```

APPENDIX B

INPUT-OUTPUT, SYNCHRONIZATION, AND ERROR DETECTION

In addition to the sequential decoding itself, the program for which is described in Appendix A, the on-site computer must have provision for initially determining the beginning of the data frame, including the correct phasing of information and parity bits and resolution of sign ambiguity; it must then be able to input data from the six-bit data buffer and output decoded data to the station tape, with a fixed two-frame delay, and with the proper relation to the decoding program. Finally, although a decoding failure due to excessive decoding computations is a fairly good indicator of which frames have errors, it is useful to have a still more sensitive error-detection criterion. The reverse decoding scheme proposed by Ames has proved to be remarkably sensitive; for the record, however, we present here another possible approach.

Timing: Initial Concept

We assume that every time the input buffer fills with six bits, an interrupt is generated. These interrupts will serve as an external clock to control the over-all timing of the program.

In the initial concept, while the decoder is decoding Frame I, the input/output interrupt program will be loading the data corresponding to Frame I+1 [DATA (I+1)] and outputting the decoded estimates for Frame I-1 [EST (I-1)]. To minimize storage and to simplify the interrupt routine, we use the same table for input and output; whenever an interrupt occurs, we first write EST (N,I-1), then read DATA (N,I+1) into the same place in the table. Thus we can get away with three 224-word tables; DATA (I+1)/ EST (I-1), DATA (I) and EST(I). Finally, to avoid moving the decoded EST table to the I/0 table at the beginning of each new frame, it is convenient to redefine the three tables for each new frame as follows:

Frame:	I	I + 1	I + 2
TABLE 1	DATA (I+1)/EST(I-1)	DATA (I+1)	EST(I+2)
TABLE 2	EST (I)	DATA (I+2)/EST (I)	DATA (I+2)
TABLE 3	DATA (I)	EST (I+1)	DATA (I+3)/EST (I+1)

This simply involves rewriting a few dozen addresses in the decoding and interrupt programs at the beginning of each new frame.

Over-all control of the decoding program is governed by the I/0 interrupt index, NI/0. After the end of a frame (NI/0 = 223), the interrupt program returns to a program BEGINF, which does the following:

1) Resets NI/0 to 0.
2) Redefines TABLE 1 - TABLE 3 as above.
3) Sets a decoding flag DFLAG to 0.
4) Initializes and starts the decoding program.

Decoding then proceeds independently of the periodically occurring interrupts. Whenever decoding is complete, DFLAG is set to 1 and the decoding program idles. When NI/0 reaches a certain number NBREAK, the interrupt program returns not to the body of the decoding program but to a program ENDF, which does the following:

1) If DFLAG=0, sets the last 14 bits of EST (I-1) to zero and the last 14 of EST(I) to the Barker sequence and mode ID. The 14 zeroes will indicate a decoding failure to the TPL.
2) Performs all the data reduction and housekeeping functions now performed by the on-site computer.

It is clear that NBREAK must be chosen small enough to allow all these functions to be completed and in any case not larger than 209. Segregating the housekeeping functions in this way maximizes the time available for decoding, and probably enables these functions to be performed most efficiently.

Timing: Overlap Concept

As we are pressed for time, it would be useful to avoid wasting the idle time when the decoder finishes a frame early. It appears that without much difficulty we can start the decoder on the next frame as soon as it finishes the current frame, if we only guard against the decoder running ahead of the input data.

In the overlap concept it is assumed that when the I/0 program begins a new frame, the decoding program is already decoding the previous frame. Thus BEGINF must be revised to do only the following:

1) Reset NI/0 to 0.

2) Redefine the I/0 table only.

3) Set DFLAG to 0.

Whenever the decoding program finishes decoding a frame, it transfers immediately to ENDFA, which

1) Sets DFLAG to 1.

2) Redefines the DATA and EST tables.

3) Performs all housekeeping functions.

4) Initializes and restarts the decoding program.

Finally, whenever NI/0 reaches NBREAK, the interrupt program checks DFLAG. If it is still 0, it then returns to ENDFB, which

1) Sets the last 14 bits of EST(I-1) and EST(I) to 0 and FS, MID.

2) Transfers to ENDFA.

It will be seen that under these rules, DFLAG will be 0 whenever the decoding program is in the frame previous to that of the I/0 program, and will be 1 whenever both are in the same frame. In the latter case, the decoding program must be inhibited from advancing beyond the input data. We recall that in our decoding program all forward moves are initiated by the instruction BRX *MOVEF, 2, which transfers to the indirect address specified by the indexed position in the MOVEF table, and then increments the index. Suppose we then write a little subprogram:

```
LOOPA    EAX -1,2
         BRX *MOVEF, 2
```

which decrements the index and then branches; suppose further that we replace the normal address in the MOVEF table by LOOPA; then until the normal address is restored, the decoding program will simply keep looping through LOOPA. Thus we insert in the interrupt program

```
if DFLAG = 1,
   TEMP ⟶ MOVEF (NI/0) if TEMP = 0 (restoring the previous address)
   MOVEF (NI/0 + 1) ⟶ TEMP (saving the next address)
   LOOPA ⟶ MOVEF (NI/0+1) (Blocking the decoder)
and set TEMP = 0 in BEGINF.
```

Since the average number of computations per frame in decoded frames will normally be much smaller than the number allowed, using the excess in the next frame ought to decrease the frame deletion rate somewhat.

Correlation Synchronization

The idea of this synchronization scheme is to look for the fixed frame synch (FS) and frame synch complement ($\overline{FS}$) words by simple correlation, using the 3-bit-quantized outputs for each bit, and accumulating over several frames. The sequential decoder is not used.

All that we know about the frame structure is that it has a period of 448 and the following form:

$$1X1X1X0X0X1X0X^{211}0X0X0X1X1X0X1X^{211}\ldots$$

Initially we do not know the proper phase, and it is clear from the symmetric form of this periodic sequence that we cannot determine phase from FS and $\overline{FS}$ alone. Thus our synchronization technique will leave us with a 180° phase ambiguity and a corresponding 224-bit framing ambiguity whose resolution we shall discuss at the end of this note.

We begin by focussing our attention on one half-frame of 224 received bits. Either FS or $\overline{FS}$ must begin within this half-frame. We define for each received bit the log likelihood ratio:

$$x = \ln \frac{\Pr(y|0)}{\Pr(y|1)}$$

where y is the quantization level received for that bit; there are thus 8 possible values of the log likelihood ratio, or 4 magnitudes and a sign, where a plus sign indicates a probable zero and a minus sign a probable one. The correlation weight of the hypothesis that the frame synch sequence begins at bit X is then

$$C(x) = -x(x) - x(x+2) - x(x+4) + x(x+6) + x(x+8) - x(x+10) + x(x+12)$$

A large magnitude of C(X), either positive or negative, indicates that it is likely that either FS or $\overline{FS}$ begins at bit X. We can now compute C(X) for all the 224 bits in a half-frame, and look for a large magnitude of C(X).

In general, however, the information in one half-frame will not be sufficient to distinguish between the real starting bit and some false ones. Therefore, we repeat the process for another 224 received bits, obtaining another set of correlation weights C'(X), which by themselves will also not be sufficient for a choice. We can combine these observations, if we add to C(X) the negative of C'(X + 224); when X is the correct beginning bit, C(X) will tend to be large and positive, say, and C'(X + 224) large and negative, or vice versa, so that, continuing this process for a number of half-frames and reversing the polarity of the correlation weight every 224 bits, a large positive or negative correlation peak will tend to build up at the correct starting bit, while the other correlation weights will tend to average to zero.

There are several distinct types of criteria we could use. First, we could look at a fixed number of half-frames, accumulating the correlation weights C(X), and at the end pick the bit with the largest absolute magnitude of C(X). Second, we could look until for some bit $|C(X)|$ passed a certain threshold T, and then choose that bit. Finally, we could keep track both of the largest $|C(X)|$ and the second largest, and choose the bit corresponding to the former only when the former exceeded the latter by some fixed amount T. The simulations we have done so far indicate that any of these schemes will work well for our parameters, but that the last has the shortest average time to synch for a fixed probability of false synch. For the latter two schemes we have two complete runs of a hundred trials, one at E/N_o=T=.8, and the other at E/N_o=T=.7 where E/N_o is the signal-to-noise ratio per transmitted bit. The former is at and the latter below any signal-to-noise ratio we are likely to go to.
A fixed choice after 5 half-frames would have resulted in no errors at .8; at .7, after 6 half-frames. Using the more flexible differential strategy, stopping whenever the greatest correlation magnitude exceeds the second greatest by more than T, where T is chosen after the fact as

the minimum value that would have resulted in no false synchs, we obtain the results of Table I, which indicate an average time to synch of less than 4.47 half-frames at $E/N_o = .7$, and less than 3.37 at .8.

TABLE I

Number of trials out of 100 in which after N half-frames synchronization was obtained, with no false synchronization.

N /E/N_o =	.7	.8
1	0	0
2	4	17
3	22	59
4	56	88
5	77	99
6	94	100
7	100	100

Implementation of Synch Scheme

We imagine that data will be entered into the decoding computer from a six-bit buffer, or two received bits at a time. We suppose that this buffer will contain a flip-flop which will indicate the correct phase and adjust incoming data accordingly, and that this flip-flop may be changed on command from the computer. Finally, we shall suppose that on computer command three bits can be inhibited from entering the buffer, so as to subsequently switch information-parity bit positions.

Then initial synch would proceed as follows: With each six-bit input the computer would update the correlation weights of two possible starting bits, storing them in one of the 224-word decoding tables. It would also keep track of the position and magnitude of the greatest and second greatest peaks to date. Whenever the former was sufficiently larger than the latter, it would command the buffer to assume the correct bit phase and i-p phase, wait for the presumed beginning of the next block, and start decoding immediately (as in the overlap timing concept). If, 224 received bits later, decoding were not sufficiently far advanced, it would then flip the buffer flip-flop and begin decoding with the alternate half-frame and reversed polarity. It ought to be given a few cycles of this procedure, in case of bad noise; if the decoder is not getting anywhere under either possibility, however, the initial synch program should be resumed.

Once synch is established, one can detect loss of synch whenever a sufficient number of consecutive frames fail to be decoded. In that case the initial synch program can be used for resynch. It is probably worth while to first try flipping the buffer flip-flop, in case the loss of synch was caused by a 180° phase change in the demodulator.

An Algorithm for Detecting Errors Made with Sequential Decoding

Normally one does not worry about undetected errors with sequential decoding, because they simply do not occur if the constraint length of the code is long enough and a tail of sufficient length is provided for resynchronization. For systems reasons, however, one or both of these conditions may not be achievable, and then the undetected error probability becomes non-negligible. This is the case in the Pioneer application, where the constraint length is limited to 25 by the word length of the decoding computer, and the tail to 14 or, at the most, 21 bits.

Under these conditions there are several methods of detecting errors. First, a cutoff may be imposed on the total number of decoding computations, either for a whole frame or for any part of a frame. This excludes data which are obviously noisy. Second, the likelihood of a decoded frame may be computed (as by recomputing the metric) and the frame or a part thereof discarded if it is too unlikely. Both these methods have been tried; they seem about equivalent, and neither is very sensitive. Their essential defect is that they try to estimate the reliability of the decoded frame by looking only at the decoded sequence itself, whereas it has been repeatedly demonstrated that the most sensitive erasure criteria are those which compare the most likely transmitted sequence with the sequences that are nearly as likely. Therefore, one desires a method of generating a number of the most likely transmitted sequences. One clever and effective method consists of using a sequential decoder to decode a frame both forward and backward, using in the latter case the last constraint length of decoded bits, including the tail, to initially fill the shift register. In this note we describe another algorithm related to the sequential decoding algorithm itself.

Assume that the sequential decoder has already decoded a frame. In retrospect we can recompute the metric along the path corresponding to the metric sequence already decided upon, as illustrated in Figure 1. The pronounced dip in the metric suggests that there may be errors in the middle of the sequence, but it has been established that the dip alone is not very sensitive as a deletion criterion. Much more sensitive would be a criterion which looks to see whether there is another path nearly as good as the decoded path, as illustrated by the dotted line in Figure 1.

How are we to find such an alternate path? We know from the structure of the sequential decoding algorithm that the decoded path was the first one found that stayed above all the thresholds indicated by the threshold path of Figure 2, and that no path stays more than the threshold separation Δ above the threshold path. It is reasonable to assume, however, that any likely alternate path will nowhere fall very far below the threshold path. Therefore, let us simply drop the threshold path by some fixed amount MSPACE, and search for any path which stays everywhere above the dropped threshold path. If we find any such path not the decoded path, then we can delete wherever there are discrepancies. More than one such path may be found.

We thus need an algorithm which will search for some path, not the decoded path, whose metric stays above a fixed threshold path. We can use something very similar to the sequential decoding algorithm itself, with the following variations:

1) The thresholds are not set up within the program itself, but are fixed from the beginning.
2) The ordering of the branches is such that the bit opposite to the one decoded originally is always the first to be examined; thus the decoded path will not be examined until all other paths

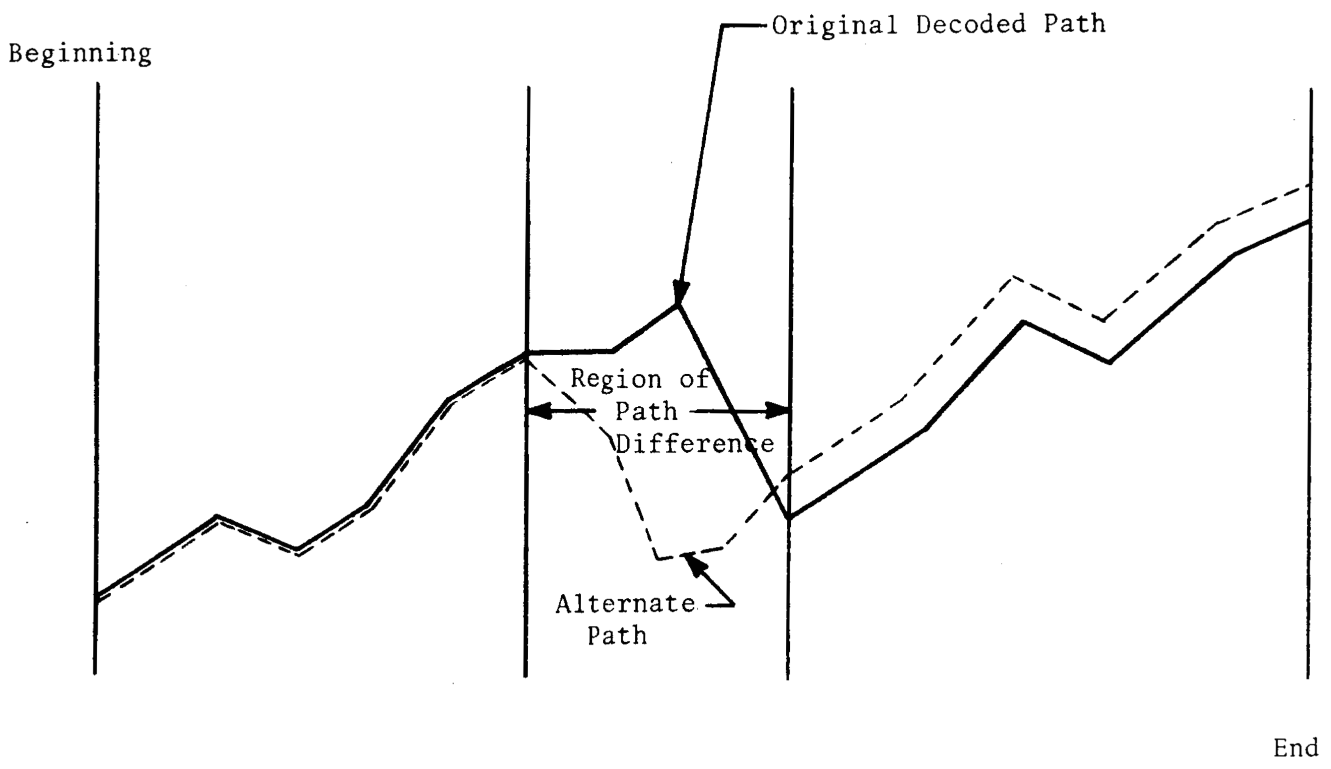


Figure B-1. Decoded and Alternate Paths

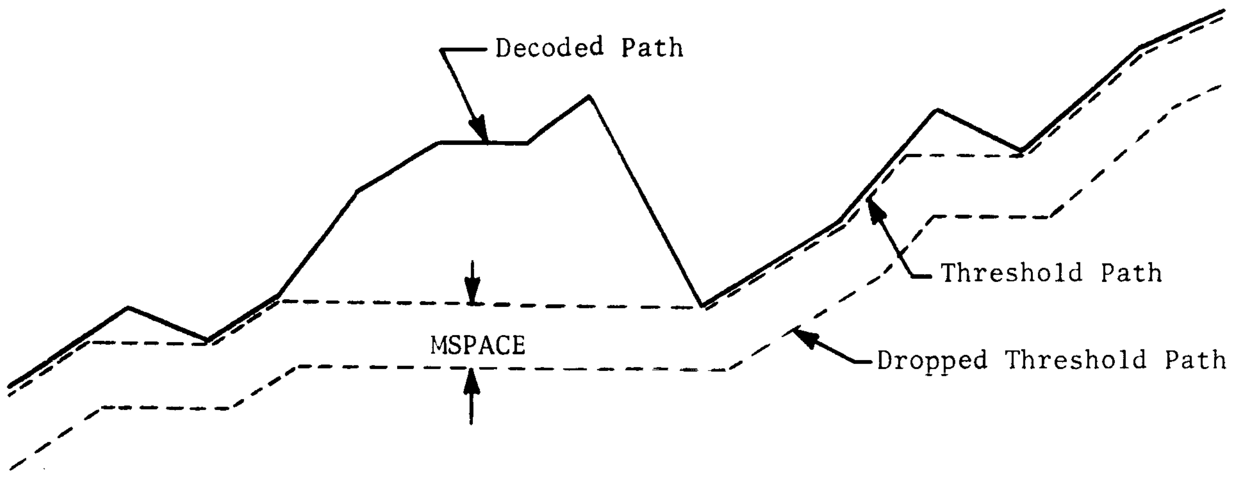


Figure B-2. Metric and Threshold Paths

have been discarded, so that we are sure to find a second path if it exists.

In Figure 3 we have drawn a word flow chart of an algorithm which resembles the Fano algorithm with the above modifications. Here, the "top branch" is the branch with information digit opposite to that on the decoded branch, and the "fixed threshold" is the value at that branch of the dropped threshold path. We recognize that certain bits may be fixed; in this case the "top branch" is the only possible branch.

This algorithm has been programmed but not yet run, so we can only estimate the probable performance. Performance is measured by effectiveness in detecting errors as against total computational time required. In the secondary search every path examined during the original sequential decoding must be examined again, though only for one threshold, not for several; this implies that with MSPACE very small the distribution of computations for the secondary search will have the same Pareto behavior as the sequential decoding itself, although with a somewhat smaller mean. For increasing MSPACE, the number of computations will probably increase exponentially as exp (K*MSPACE), where K is some unknown constant. The effectiveness of this scheme in detecting errors is unknown.

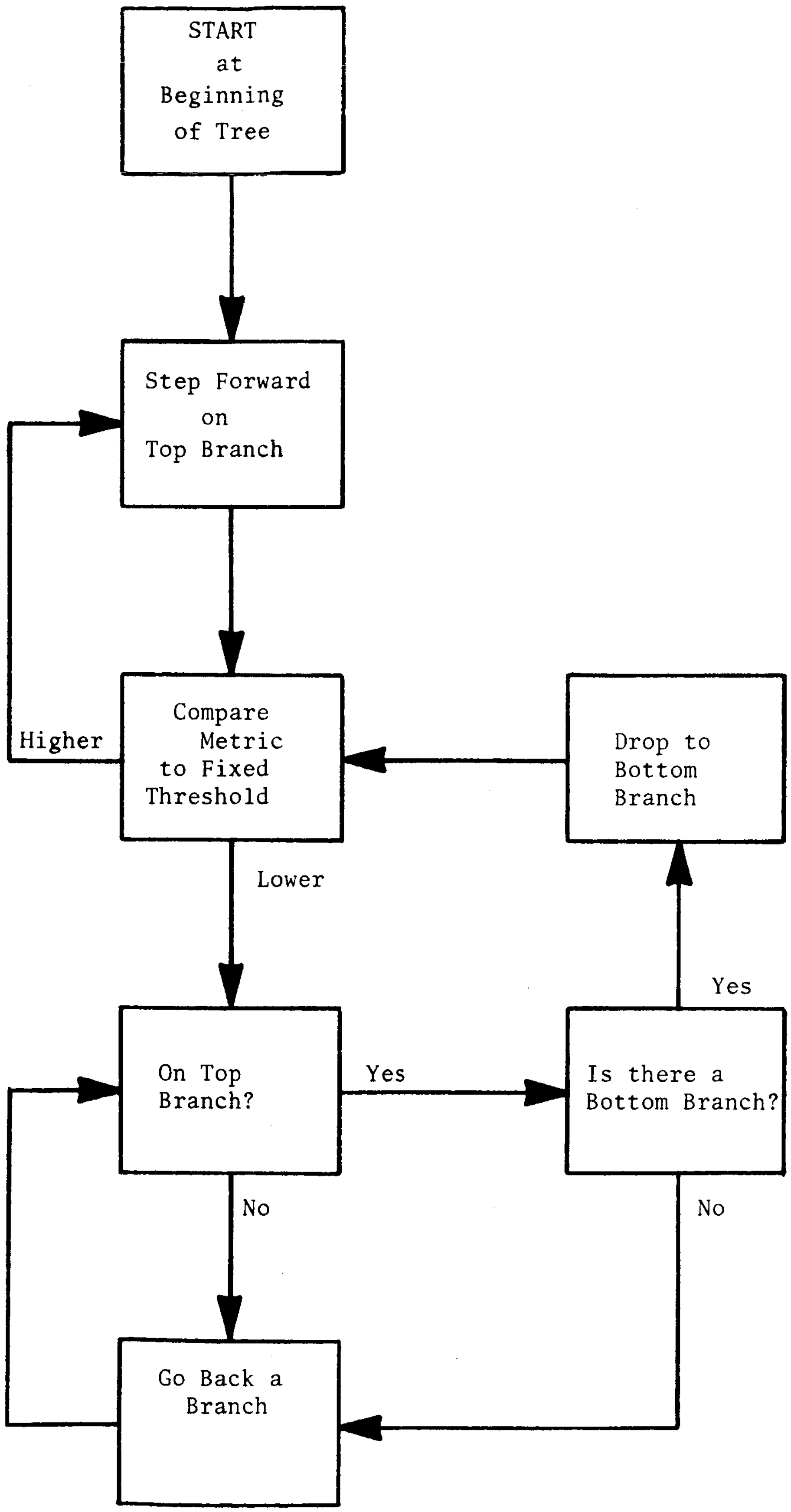

START at Beginning of Tree
Step Forward on Top Branch
Compare Metric to Fixed Threshold
Higher
Lower
Drop to Bottom Branch
On Top Branch?
Yes
Is there a Bottom Branch?
Yes
No
No
Go Back a Branch

Figure B-3.
Word Flow Chart for Secondary Search Algorithm

APPENDIX C

EVALUATION OF SMALL COMPUTERS AS SEQUENTIAL DECODERS

Sequential decoding may be performed either by a special-purpose digital device, such as in Appendix D, or by a general-purpose computer programmed with a sequential decoding algorithm. In general the special-purpose device will be much faster and perhaps somewhat less expensive than a small general-purpose computer. However, the flexibility of the computer allows it to be easily reprogrammed for different missions, or for different tasks altogether; it can also be easily modified during a mission to make use of measured noise statistics, experiment reprogramming or shutdown, or any other information which can assist it in decoding. Finally, the general-purpose computer can be used for quick-look processing, formatting and tagging, and other real-time functions involving the actual received data. It is therefore of interest to examine the performance obtainable with any of a number of small scientific computers in the $30,000-cost, 1-microsecond-speed range.

The ground rules of the comparison are as follows. Ample storage is assumed available, so that wherever time can be saved at the expense of additional storage, as by writing similar parts of the program many times or using tables rather than computing, the choice is always made to save time. Furthermore, it is assumed that the number of decoder computations per decoded bit is high enough (15 or more, say) that time devoted to input/output and any small amount of associated computation is essentially negligible. Thus the only significant quantity is the average time required for a decoder computation.

The structure of all programs was assumed to be the same as the structure of the SDS 910-920 programs described in Appendix A. That is, each decoder computation consists of an examination of the metric associated with the current branch, a decision whether to go forward or back in the tree, according to the Gallager algorithm, and an updating of the various program variables based on this decision. The transfer from the end of one move to the beginning of the next is accomplished by an indirectly addressed branch instruction, where the address comes from one of two tables (for forward and backward moves) which catalog the appropriate move for each bit in a frame; thus, although no special bits (fixed, parity, etc.) were assumed in this comparison, the framework of the program is such as to easily accommodate them. The four possible metric increments for each branch (for the information-parity hypotheses of 00, 01, 10, and 11) are assumed stored for each bit in the frame in four frame-length tables. Finally, as is implied above, the rate of the code is assumed to be one half, and the constraint length is assumed to be two computer word lengths plus one or less, or 33 bits for most of these machines.

Flow charts for the three typical moves--forward, backward on 0, and backward on 1--are given in Figures 1 - 3. In these charts, M is the accumulated metric, P the parity word, N the index of the current bit in frame, Δ the metric spacing, M00 the metric increment of the current branch given a 00 hypothesis, A00 the backward entry address for a choice of 00, and so forth. The instruction M = [M] means to reduce M by Δ if it exceeds Δ , and thus repeatedly until it is less than Δ ; ADD TAP 1 to P means to add the impulse response of the encoder to the parity word, mod 2 (exclusive or). The total number of paths from top to bottom is eleven, of which ten are distinct, and these have been labeled at the exit point by PF0, PF1, etc. From simulation results, we know the approximate frequency of each of these moves; Table 1 reports the distributions obtained during a very long search, which is typical of times when the computational load is high, and during 3000 frames in which the average number of decoder computations per decoded bit was 4.

The machines compared were the SDS Sigma 2, the DEC PDP-9, the Honeywell 3C DDP-516, and the Data Machines DATA/620. Requests to the manufacturers of the EMR Advance 6130, the Interstate IEC-1010, the Scientific Control SCC-655, and the Systems Engineering Labs SEL-810A, which appear to be competitive machines, failed to elicit enough information to allow comparison. All are 16-bit machines, except for the

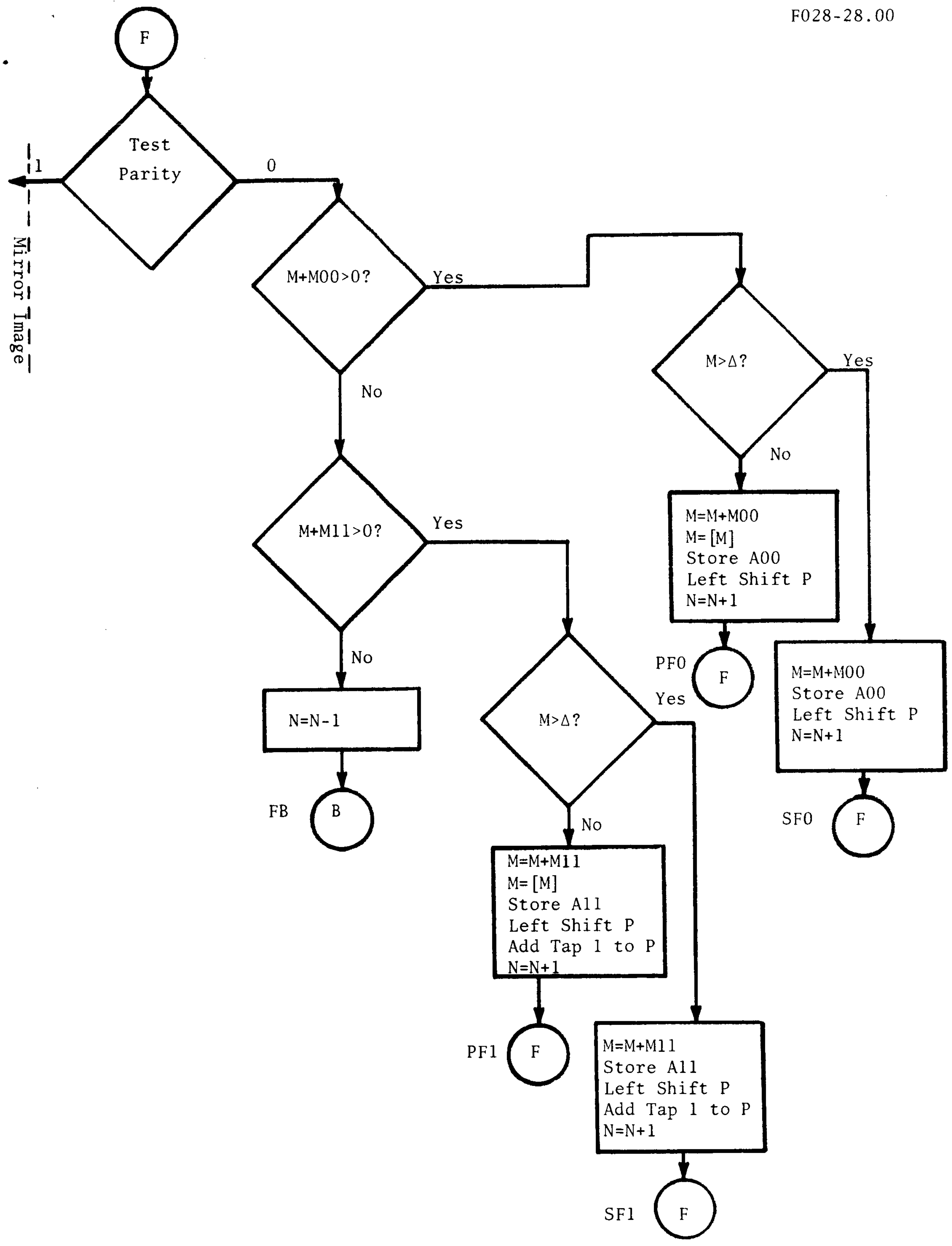


Figure C-1. Forward Moves

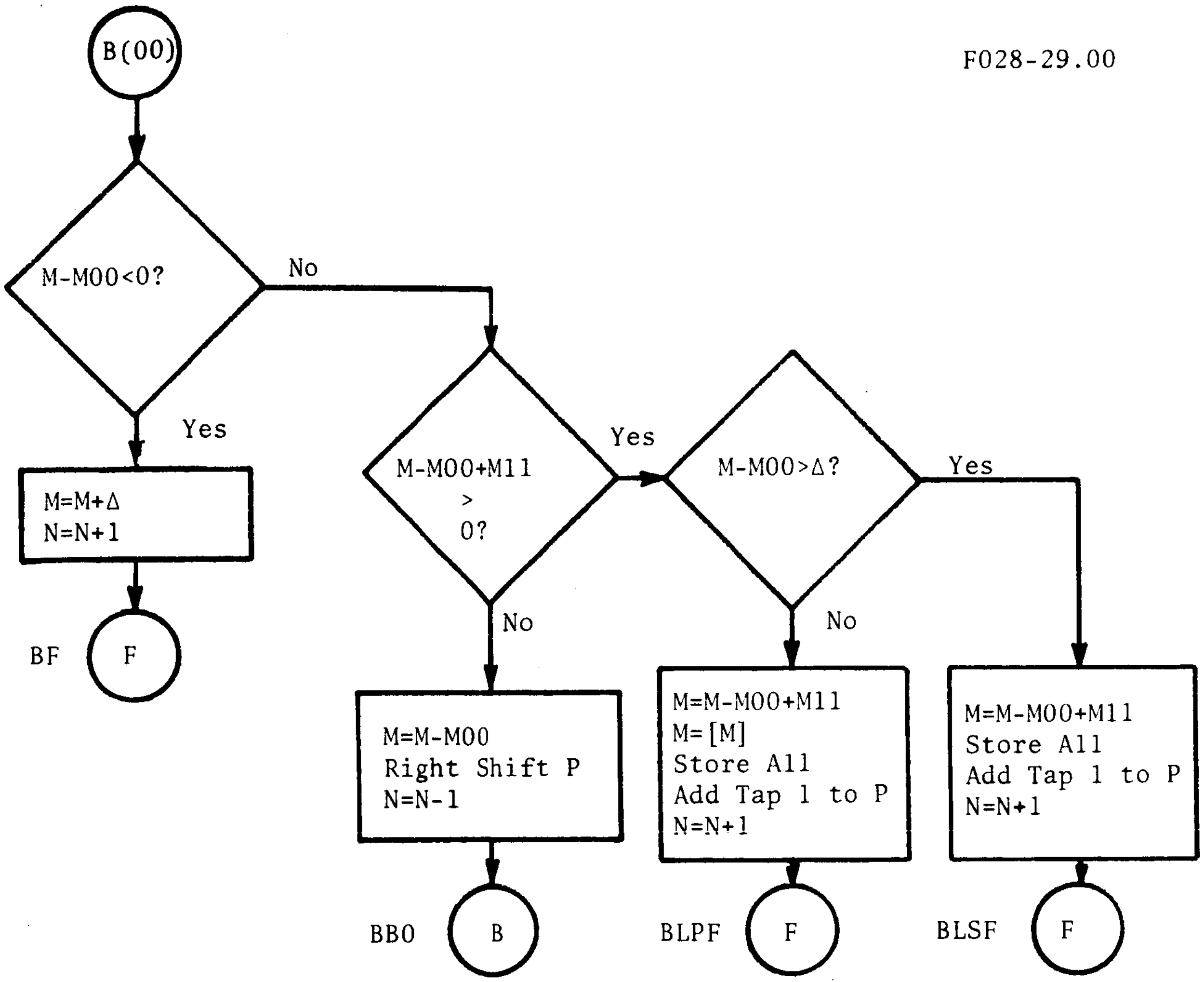


Figure C-2. Typical Backward Move to Zero Branch

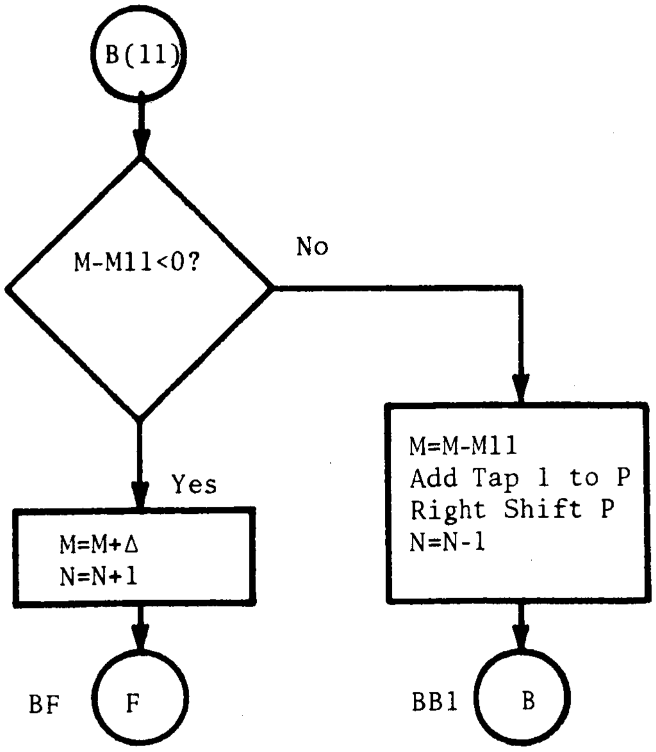


Figure C-3. Typical Backward Move to One Branch

PDP-9, which uses an 18-bit word; all have a basic cost in the vicinity of $30,000. Sequential decoding programs were written for each of these machines. There was a substantial attempt to organize the programs for the most efficient use of each machine's peculiar quirks, so the internal organizations of these programs, though in general following the flow charts of Figures 1 - 3, were often quite different. Optimization was not pressed to the limit, but we would be surprised to find a program more than a few microseconds faster than any reported here. Table II gives the average time for a single decoder computation on each of these machines, where the frequencies of the different move types were obtained from the "long search" column of Table I. This table also includes comparable times for some other machines we have written programs for, for comparison and general interest.

TABLE I

Frequency of Different Types of Moves (Percent)

	Long Search	3000-Frame Average (4 comp/bit)
PF0	12.6	16.8
PF1	9.3	14.7
SF0	12.6	12.8
SF1	5.1	5.4
FB	11.0	10.0
BF	1.5	2.5
BLPF	1.7	1.3
BLSF	7.6	6.1
BB0	15.6	12.1
BB1	23.2	18.2

TABLE II

Average Time for a Decoder Computation, in Microseconds

Machine	Time
DDP-516	26.5
Sigma 2	30.0
PDP-9	49.2
DATA/620	55.4
IBM 360/75H	9.1
IBM 7094	31.0
SDS 930	50.3
SDS 910-920	201

APPENDIX D

STUDY OF A SPECIAL-PURPOSE SEQUENTIAL DECODER

In this Appendix we report the results of a preliminary design study of a special-purpose sequential decoder capable of faster operation than a small general-purpose computer, and quite sufficient for the on-line needs of the Pioneer program, but by no means designed to do the ultimate in sequential decoding.

The basic parameters of the design are matched to the Pioneer requirements: code rate 1/2, constraint length about 25, 3-bit quantization of the received likelihood ratios, and provision for operating with a data frame structure of the order of 224 bits long, with fixed words in the frame and with a simple parity check on seven-bit variable words within the frame. With these parameters, and using currently available components, it appears quite feasible to build a machine which will perform a decoding computation in 1 microsecond. This would permit decoding 512 bps data completely in real time with no appreciable degradation due to computation speed. (Even 100,000 bps data could be handled with some degradation.) Such a machine would imply a materials cost of about $10,000 and would occupy about 11 inches in a standard relay rack.

It should be noted that several of the design parameters are quite flexible, as will be seen from the detailed discussions below. Constraint length of the code is variable at negligible cost. Code rate could be decreased rather easily, with little change in the basic organization of the machine, although at a noticeable cost. The degree of input quantization could be changed with minor design changes; chiefly an expansion of the word length in the core memory, an increase in metric table sizes and perhaps some additional constraints on the table design. Frame length and format can also be changed without any major effect on the design.

Three speed regimes were considered in the study: slow machines, with computation speeds of 10 us or more; a medium-speed machine with computation speed of the order of a microsecond; and an Interim Ultimate Machine, with the fastest possible computation speed, perhaps 100 nanoseconds. These speed breaks are roughly determined by breaks in the speed of components: slow decoders by classic 6-10 us core memories and discrete component logic; medium by newer 0.6-1.5 us core memories and the most common integrated circuit logic; IUM by the newest devices and those which continued to be announced during the study, i.e., scratchpad memories, plated wire memories, multi-aperture memories, ECL logic, fastest TTL logic, etc.

The slow decoder was adjudged to be properly the field for software on mass-produced general-purpose computers, and not further considered. Such decoding has been discussed earlier in this report. Virtually the entire effort was placed on medium-speed decoders, where the study would carry a high credibility (and validity); IUM effort was devoted to consideration of direction of design effort, subdivision to faster and slower sections, component surveys and follow-up, and so forth.

In the following we shall first consider the structure of the medium-speed decoder in terms of its major components, then discuss these components and their functions in more detail, down to the level of circuit diagrams where appropriate, and present a tabulation of materials involved in its construction. Finally, we shall discuss the problem of building much faster decoders.

Over-all Structure and Characteristics of the Decoder

In Appendix A of this report, as well as in earlier reports, we have discussed the nature of sequential decoding, and reference to those discussions will provide an appropriate background to the following discussion where necessary.

A rate 1/2 sequential decoder operates by making tentative decoding decisions, one by one, as it progresses through the data. By keeping track of the likelihood of its over-all decision path, it can tell when it has probably made a false decision and it then backtracks, as far as necessary, to find better paths. Thus its amount of computation varies, depending upon the amount of backtracking it must do. In order to avoid hanging up on an overwhelming number of trials, when some particularly bad received pattern is being decoded, we utilize the frame structure of the data. Each new frame (or some integral number of frames), the encoder is reset to a known pattern (e.g., all zeroes) and the decoder can start afresh at that point. If too many computations take place, so that a frame is not decoded fast enough, the decoder simply gives up, indicates a nondecoded frame, and starts afresh on the next frame.

At a decoding speed of 10^6 computations per second, with a data rate of 512 bps, there is an average of 2000 computations per decoded bit available. This is enough to make the probability of decoding failure very small at reasonable operating levels.

The major components of the decoder are shown in Figure 1.00. We here describe their over-all functions. How they are realized in detail will be discussed later.

The memory receives input data from the receiver as six-bit words; three bits for the information bit likelihood measure, and three for the parity bit. The memory also stores the decoder decisions, and they are passed out from the memory to the outside world at the same speed as input is received, but with a fixed delay equal to the number of bauds in the memory and the replica encoder.

A faster memory is used as a buffer to store three words of data at any given time, one the word being worked on, one the previous word and one the next word. These words are shifted and exchanged for fresh ones only at the basic 1 usec cycle of the core memory. However, access to the current word is available almost immediately. Thus, computation can start without waiting for the comparatively long access time of the core memory.

The Control Logic box can be thought of as dominating the decoder. For a decoding computation it makes a hypothesis as to the information bit, and then feeds that hypothesis to a duplicate of the encoder which, using previous hypotheses as well, computes the parity associated with the information hypothesis. The hypothesis, corresponding parity, and data word (six bits) are then used to enter a Table Look-up which puts out the metric increment corresponding to its inputs. This enters a Fast Adder,

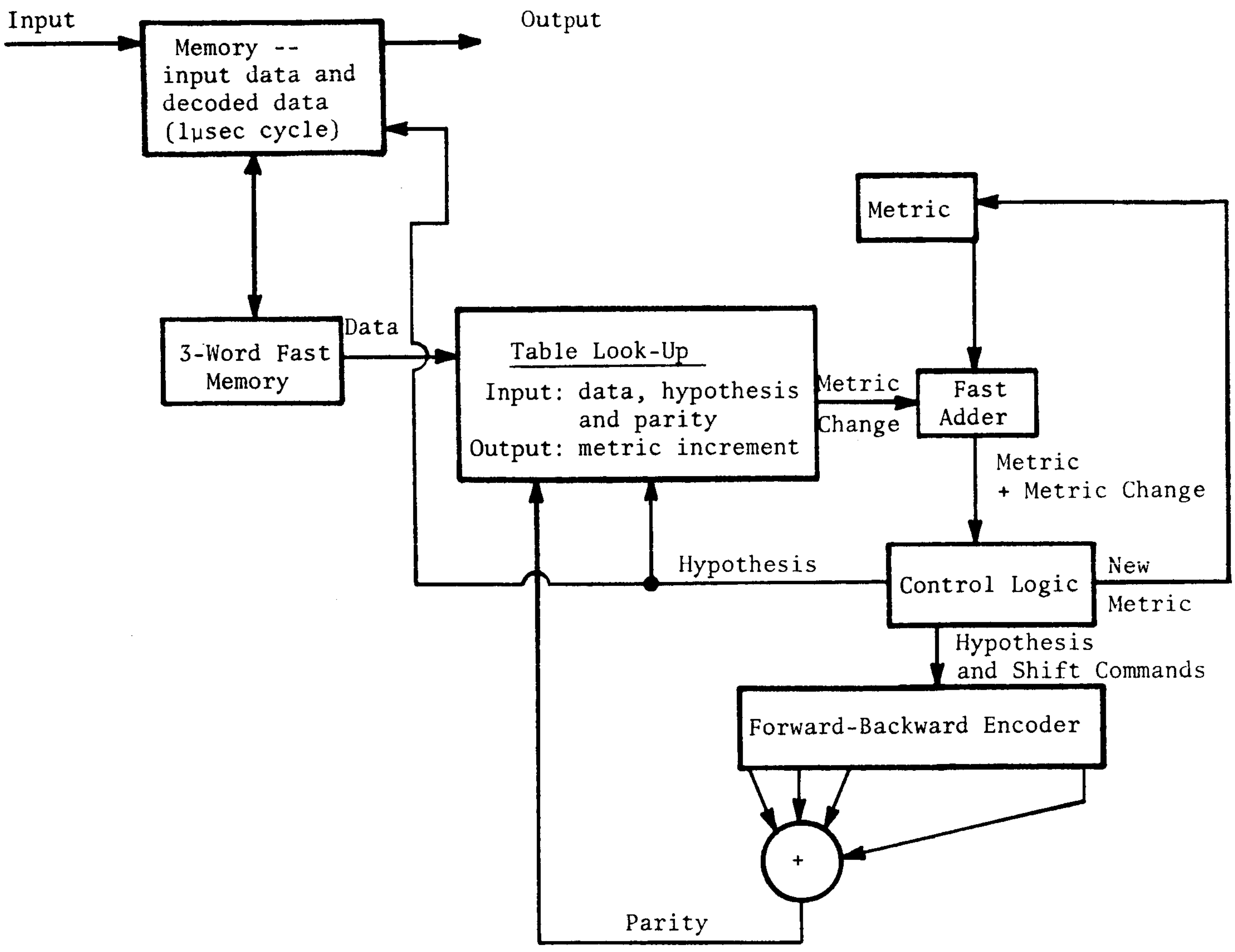


Figure 1.00 Overall Decoder Structure

the other input being the Metric (total metric from previous decisions); the output goes to the Control Logic. Depending upon the value of the metric plus metric change, the Control Logic decodes to move laterally (try the alternate information hypothesis for that bit), move ahead, or move backward, in its trials.

The over-all sequence of operations of these elements can be seen by considering Figure 1.00 in conjunction with Figure 2.00, which is a condensed version of the decoding algorithm. (Figure 10.00 gives a more detailed flow chart of the algorithm.) The language used here to follow the algorithm treats separately each logical function. In the hardware machine, however, no special command is needed to form parity, for example, or test H=0.

We start by considering a forward move, entering Figure 2.00 at the point labelled F. The first step is to hypothesize a zero (H=0) for the information bit and compute the metric change this implies. The Control Logic makes the hypothesis, instructs the replica encoder to compute parity corresponding to a forward move with zero information bit, and feeds the hypothesis for information, the parity, and the corresponding likelihood ratios in the data word into the Table Look-up. The output is the metric change, and the adder forms the sum of the old Metric with the change (A = M + M(DHP) in Figure 2.00, where M(DHP) means the metric change corresponding to the Data, Hypothesis and Parity).

If the adder output is positive, the hypothesis is accepted and we move forward. This involves putting the adder output in the Metric box as the new metric, right-shifting the hypothesized bit in the encoder to be ready for the next hypothesis, advancing the counter (N) which tells where we are, and calling for the new data D(N) corresponding to the next word.

If the adder output is negative, we try a lateral move, hypothesizing an information one. The box in the algorithm labelled H=0 tests that we have just tried a zero. Since we have, we move upward on the figure, hypothesizing a one, inverting parity to fit the hypothesis, and again computing the metric change and testing it for sign. If it is positive, we are happy and move forward, as before.

If A had been negative when we tried H=1, we could not accept the hypothesis and must move backward. Checking that H=0, we move back by left-shifting the encoder to bring the previous decoded bit to the front, recovering the parity associated with it, decreasing the N counter by one and calling for D(N), the previous data word. We are then at Point B in the algorithm.

We compute the metric change corresponding to the DHP available (this was our previously used metric increment when we moved forward in the past), subtract it from the current Metric, and examine the sign of the result. If the result is positive, we must try a lateral move if possible. The new metric is entered in Metric, and we return to the H=0 question in the algorithm. If the previous decoding decision was a zero, we will now try a lateral move, as before. If the previous decision was a one, meaning the lateral has been tried already, we will move backward again.

If, on a backward move we find that the recomputed metric was negative, we add Δ to the metric and move forward. This involves adding to the output of the Adder (the recomputed metric) and putting the result in Metric, right-shifting the encoder to compute parity for the forward move, advancing the N counter and calling for data D(N) and re-entering the forward move at F.

We have so far not mentioned how we take account of constraints on the message, such as known bits within a frame of data, or a simple parity constraint, such as a seventh bit parity sum on each six bit word. These constraints are easily taken care of in the Control Logic which tracks the constraints by their known position in the frame. Fixed bits simply mean that at a node where those bits are being decoded, the information bit is forced to the known value. The operation is essentially the same as that discussed in Appendix A. Instead of trying both "zero" and "one" hypotheses, we use only the known value. A parity bit is similarly forced by computing the sum of the decoded bits involved in it and using the result as the single hypothesis permitted at the corresponding decoding node.

Design Considerations for a Decoder

In the following section we explain some of the most critical problems that would arise in the course of the design of the sequential decoder and explain some solutions to them. The solutions are worked out in varying degrees of detail as necessary to show that the difficulty is clearly solved. Where the practicality of a particular section of the machine is involved, that section is designed and timing and wiring shown for a commercial set of logic modules. The design of an actual decoder to meet a detailed specification is thus brought closer to the routine problems of digital design.

The major effort is applied to the critical elements of the medium-speed machine of approximately 1 microsecond per node decision. For ease of explanation this is first handled as a decoder taking a variable time slightly over 1 microsecond on the average; this is implied in Figure 13.00. There follow the specific complications needed to achieve a time of exactly 1 microsecond.

Most of these considerations apply with equal force to the Interim Ultimate Machine. However, the last section considers some of its peculiar problems which are in general solvable by minor development rather than off-the-shelf components.

The Encoder

The encoder consists of a shift register with certain bit positions connected to a mod 2 adder to compute a parity bit, as shown in Figure 4.00. The incoming information is split up, being sent out on the channel as well as down the shift register. For each channel information bit, the current state of the parity is sent out as the parity bit. At the beginning of a coding block the shift register is forcibly set to a fixed pattern which may as well be all zeroes. This does not disrupt the flow of channel information bits or interrupt the parity flow, but that first parity bit is computed as though the current bit were what it is but all previous bits as far back as the register can hold them had been zero. The coding block structure is thus not readily visible in the stream. For convenience in.locating the coding block structure, it is made coincident with the information block structure, which is of a suitable length. When the decoder tackles a branch at the beginning of an information block, it sets its own history register (of past hypotheses) to all zeroes; when it is forced back to that point in a search for good matching, it does the same thing. Whatever else it may do about that back search, it obviously never carries back beyond the first bit of the information block.

Decoding Action

A compact form of the decoding operation is shown in Figure 2.00. This is closely related to the operation of the machine, and may be used to follow the action in this description. The location of the data and the computation results may be followed in the memory-encoder relationship shown in Figure 13.00. Here the memory is shown in ring form. One should visualize the ring as being stationary; an address number is assigned permanently to each location around the ring, which has six bits for storage of the channel data and one bit for storage of a hypothesized bit of the output stream.

Access to the ring is of two kinds, the slow channel access and the fast node access. The channel input arrows indicate a slow regular progression around the ring (clockwise in the drawing). For each channel information baud, the access clears out the six old bits left from the last passage of the channel access and writes in the six fresh bits. At the same time the output arrow reads out the hypothesis bit which had been left there for it by the node circuit and wipes it off the memory. Since one memory operation is used for both actions, the operation in the conventional core memory is of the type known as READ-MODIFY-WRITE, so that we read out 7 bits, losing the six old channel bits and sending the hypothesis bit to the decoder output; then write in an unrelated seven bit word consisting of the six fresh channel bits and a zero for the hypothesis. The duration of this "slow" operation is actually as fast as the fast operations, about 1 usec. At its conclusion the memory is turned over to the node circuit and will not be required until the next channel bit entry time. A channel address counter is updated one count, ready for the next channel access, represented by the channel arrows on Figure 13.00 jumping one step clockwise. Thus, the channel access slowly and steadily progresses around the ring leaving a trail of channel data and picking up hypothesized information bits prepared since the last revolution.

Between these brief channel operations the node mechanism has control of the memory and has been swinging about the ring jittering in either direction. The six channel bits are consulted but not altered. A copy of the encoder capable of two-way operation lays in the hypothesis when going forward (clockwise); since this comes off the end of the register, the length of the register is in effect added to the size of the hypothesis memory which is thus longer than the channel memory by the twenty-seven bits in the register. (This determines the constant time delay at the channel access between channel input and corresponding hypothesis output.) When progressing backward, the H (hypothesis) bit is fed into the end of the register to reconstruct the past states of register history. The cycle for forward operation is: READ seven bits out of memory, ignoring H and storing the six D (channel) bits in six latch circuits: WRITE seven bits consisting of the oldest H bit in the register and the six D bits in the latch back into D. For reverse operation: READ H into the end of the register and D into the latches: WRITE zero into H and the six D bits back into D.

A separate node-address counter is used to control this access; it is counted up or down after each branch operation as the algorithm determines the direction of the next node move. This counter is independent of the channel-address counter except that coincidence of the address indicates overflow. (A possible second exception occurs in the block tracker if track is furnished externally; see the account of the tracker.)

Memory Size Considerations

The utilization of the memory by the channel speed circuits and the node speed circuits is shown in Figure 11.00. Here starting from a common point (the most favorable condition, with node caught up to the channel) the node circuit jitters back and forth over a bad section, reaching its maximum depth of backsearch at a time when the plodding forward progress of the channel circuit has brought it around to the same point. If we take as practical figures a channel rate of 512 bps, an allowable backsearch of 50 branches involving 500,000 separate branching decisions at 1 usec per decision, then the node circuit will run back 50 addresses in 500,000 useconds per baud, 250 fresh addresses. This unbalanced allocation of memory capacity has caused these two operations to be called Hare and Tortoise in the internal project discussions.

From the simulations a memory size of 256 words was judged adequate. Analyses made from the viewpoint of buffering blocks (on the common idea of averaging out the effects of blocks which contain long searches) reveal that addition of large amounts of memory yield almost no gain in relaxation of node speed for a fixed rate of occurrence of overflow ("discard rate"). The effect may be viewed as increasing the machine's exposure to such searches. Presumably the blockless view of Figure 11.00 will meet the same effect. This leaves us with a core memory of 256 words of 7 bits each (probably 8 bits, to the nearest standard core size) with a READ-MODIFY-WRITE cycle time of 1 usec.

Tracking

Acquisition and tracking of bit sync is done on the channel stream prior to the decoder (as a whole) by the receiver circuitry which needs it to quantize the output. This bit sync is assumed to be furnished to the decoder, or if not, is trivially recovered.

Block sync is not utilized by the channel speed section of the decoder, which operates as a transparent fixed delay. Block sync is utilized by the node speed circuitry where it is necessary for coding recovery and helpful for forcing character parity onto the convolutional parity estimates. Information from which block sync can be recovered is available at both channel speed and node speed circuits. Hence, the block tracker is operated at node-speed in the form of a counter updated together with the node-address counter.

If block sync is recovered by the receiver, it will be handed to the decoder at channel speed. It will be remembered that the two speed regimes are connected only at overflow time. Initial block sync could then be handed over to the node speed circuit by maintaining the overflow coincidence (blocking backward travel for the node circuit) until the first block start is encountered.

Memory Tail

When the node circuitry is lagging almost to the point of overflow, using channel bits laid down long ago and putting out hypothesized information bits just before the channel circuitry sucks them up, these information bits are comparatively uncertain since the node circuitry may decide it wants to do a backsearch on them. One might visualize a rule that no bit be put out by the decoder which is not at least 50 bits behind the node circuit at the time it is put out. This could be implemented by a constant delay shift register at channel speed added to the channel speed memory output. Logically this is unnecessary since any particular bit is simply irrevocably committed to the shift register. An equivalent action is to omit the shift register and tag the rest of the block in which an overflow occurs. A suspect bit put out just in advance of the node circuitry will find in the subsequent history either that the node circuit will leave the vicinity and run on forward, in which case the suspicion is disproved, or that the node circuit will in fact overflow. The decision to use a shift register tail depends only on whether the subsequent data handling can buffer a block to see if it is finally tagged for an overflow.

Node Interrupt

The core memory has a single access point which must be made available to two circuits each steadily clocked at different rates. Designs were considered for a fixed ratio, in which bit tracking at channel rate would alter a common clock, preserving the count of a channel access every 2000 node accesses (for 512 baud input); and also for an alternative asynchronous interface in which a channel demand for access would momentarily interrupt the node access circuits. The second method would have the advantage that the received baud rate need not be specified to the decoder. However, this was not considered a large advantage in the expected operation, when even for very long delays in response of a very remote vehicle to commands to change data rate the time of response is accurately known at the ground station. The method thought to be best for simplicity of design with integrated digital circuitry was to run off a fixed count of node accesses and have node count interrupt itself. The incoming channel access would allow another cycle of node accesses.

Timing Within Node Circuits

Within the node circuitry a processor must extract data from the memory, come to a decision about the current information bit and decide whether to ask next time for the memory address one count forward or one count backward from the one last consulted, all in time to be ready for the next access. The circuitry for this processor is the common integrated circuit digital logic often referred to as 5-megacycle circuitry. This implies that flip flops can change and settle through a few levels of gating in time for another flip flop clock pulse 200 nanoseconds later. A typical action in the forward direction can be followed from Figure 2.00. After a forward access the encoder section hypothesized a zero for I (information) and finds the associated P (for parity) based on the history of I. When it has been given the three-bit data on the Channel I, it computes from that and the hypothesized I the corresponding I metric. Similarly it finds a metric for P. The sum of these is the metric M for the branch, shown in Figure 2.00 as M (DHP). It finds these metrics from a look-up table in the form of a fixed network yielding the several bits representing M when excited with D_I and H or D_P and P. The resulting M (DHP) it then adds to the cumulative metric stored in a register to yield the provisional sum, A. The test is now made on the polarity of A. If the sum is still positive, then the zero is a successful guess for I; the provisional sum A is declared to be the new value of M, the next move is called for forward, the encoder register is shifted right for the next P to be prepared, the node address counter is clocked one count up. If the value of A is so large as to exceed Δ , the Δ is subtracted from A. This is accomplished simply by restricting the value of Δ to a binary number whose lowest digits are all zeroes, so that a normalized value of A is obtained by forcing the highest digits of A to zero. If the polarity of A was negative, then a trial is made from scratch with the value of I at one, and the corresponding P. To save time in determining P, the circuit takes the opposite of the first P. Thus I and P enter the table through exclusive-or reversing circuits, operated by a flip flop whose significance is Normal or Reversed polarity. If this reversed polarity is successful, the I shift register picks up the reversed I for the second stage of the register to preserve the result of the decision.

This examination of a node cycle reveals two major contributors to the lapsed time, the look-up table and the adder. The design of these was carried out in the midst of a rapidly changing technology of readily available material, as both of these functions are common bottlenecks in digital systems and component manufacturers are busy in these areas. (The design of the medium-speed sequential decoder is predicated on the use of readily available components of readily predictable behavior.)

Size of the Metric

The speed of the table and of the adder depend heavily on the size of the metric on which they must work. This size was derived from figures being used midway in the study; later considerations may reduce this slightly. The accumulated metric, M, is the sum of many individual branch metrics ΔM. The latter function is made up of ΔM_I plus ΔM_P for any branch. These components run from +1/2 for a good agreement to -16 for total disagreement between hypothesis and channel sample. Hence, ΔM ranges from +1 to -32. The resolution of ΔM is determined by the smallest increment used for ambiguous levels near zero; it was determined that this region from zero to +1 be taken to the nearest 1/16, which requires 4 places after the binary point (analogous to decimal point). The range of ΔM thus requires 4 places after the point, 5 places after it, plus a sign bit, for a total of 10 places.

The accumulated M does not grow over the whole block, for in the present algorithm (Figure 2.00) we subtract Δ from a metric which has successfully passed Δ in the forward direction. Here Δ is taken as

16, although flexibility for changing Δ is retained in the design. However, if the node circuit finds a decreasing M leading to a valley dipping below zero, it retreats back over the hill until it senses a negative region behind it, adds Δ to the M at that point and runs over the same ground, now increased by Δ . If it still cannot cross the valley on positive levels, the process is repeated. These back and forth searches occupy so much time that an allowance of 50 nodes back was set for all factors affected by them. The maximum size of M is thus set by the size of the hill one can elevate by this method in a back search of 50 steps. Ten bits before the binary point were allotted for this, and of course it must retain 4 bits of resolution after the binary point. The sign bit is a 15th bit but this is only an implied bit, as M is always positive. However, the adder must use a 15th bit, as ΔM has a sign bit. Hence, we need a 15 place adder and a 10 bit table. The speed of these elements was set as a reasonable fraction of a 1 us cycle by requiring that each of them finish its operation within 200 nanoseconds in expectation that a basic 5 mc/s clock will be used.

Numerical Notation

The value of ΔM may be positive or negative. These values are added to M during forward progress and subtracted from M to retrace our steps. To aid the easy changeover from addition to subtraction of values of both polarities, we adapted the two's complement method of assigning values in which negative and positive numbers form a continuous series, swept from most negative to most positive by adding a fixed increment:

	Sign and Absolute Value			Two's Complement		
+ 1	0	00000	00001.0000	0	00000	00001.0000
0	0	00000	00000.0000	0	00000	00000.0000
- 1	1	00000	00001.0000	1	11111	11111.0000
-32	1	00001	00000.0000	1	11111	00000.0000

A positive metric can be made negative or a negative made positive for subtraction on a reverse path by inverting all the digits and adding the smallest increment, .0001. This process would normally take as much time as a full addition, since the carries might propagate the full length of the metric. Hence, when a metric is to be subtracted, it is used in the one's complement for M, each digit being taken as inverse. However, normal addition rules do not hold for this form. Therefore, subtracted metrics are accompanied by a carry input to the first adder stage, just as carries are used between stages; any propagation of carries runs concurrently with those of the main addition.

The Table

The input table could be built very simply if it were built to accept a 4 bit input (3 bits of D_I and 1 bit of I) and put out a 10 bit ΔM_I; entrust this to the adder; go back for the 4 bits of D_P and P, and put out 10 bits of ΔM_P; and let the adder add them. This would lead to excessive delay in this addition prior to the trial addition of M, or to excessive complexity in the adder by making it accept M, ΔM_I, and ΔM_P simultaneously. For this reason it was decided to run the table as one large table accepting D_I, D_P, I and P and yield the 10 bit over-all ΔM immediately. We examined the table in some detail, as with all elements which could be foreseen to lie on the critical time path.

As it became obvious that high fan-out and fan-in were important elements of the table, we did the first tries with DTL logic rather than the cheaper RTL. The characteristics of the Fairchild 930 series were assumed. To avoid the controversy on the reliability of the DIP package, still raging now as strongly as at the beginning of the study, we restricted ourselves to forms available in the commercial temperature range hermetically sealed flat packs. The line is usually quoted as leaving 30 nanoseconds maximum delay per gate, with a fan-out of 8.

(Several varieties of Read-Only Memories were investigated as an alternative for this table but were too expensive, too slow, or too special. For the medium-speed decoder rapid availability and predictability of the components was considered primary. Among these forms were core memory, integrated diode blocks, MOSFET arrays, multiaperture memories, plated wire memories, and printed circuit induction. Some of these were considered more favorably for the main memory of the Ultimate Machine which was examined in less detail.)

Several methods of organizing the memory were considered and sketched out. If we take an 8-pair input (8 bits plus the inverses), develop 256 intermediate lines for the possible varieties of input, and fan these into 10 output lines, we have a very fast device (3 gate levels) but with illegal amounts of fan-out (heavy load) and fan-in. If we subdivide this into two halves, each of four bits (one for I and one for P) yielding 16 variety lines each, then combine these in 256 2-input gates (each variety line running 16 inputs by being split into 2 I or P line drivers) of inverse polarity, then gather these into 16-input NOR gates for inverse polarity (normal NANDS) which are double-inverted into another rack of NOR gates feeding the ten output lines, we achieve a structure at once large and slow. In fact, the material would equal the cost of a core memory.

Organization by coincidence on a 16 x 16 matrix would require a gate per intersection, with large size and illegal loading. This organization is that of core memories and reminds us that core memories are feasible in that organization only because of the small cost and interaction of the element at the intersection.

The final solution took advantage of the individual nature of the table. There are 8 varieties of D_I, each matched to 2 values of I to yield 16 varieties of ΔM_I. However, the mirror symmetric value of D_I when associated with the opposite assumption of I, yields the same metric. Hence, we can combine at one point (one multi-input gate) a value of D_I and I, and the value of the opposite D_I and the opposite I. Moreover, having done the same for P, we find that the final metric for a high likelihood I and low likelihood P is identical with the metric for the I and P in exchanged likelihoods, rather like a commutative rule. The influence of fan-out limitation even at 8 is seen by the resultant structure; three D_I bit pairs run 8 NAND gates for D_I, while three D_P bit pairs run 8 NAND gates for D_P; each of these 16 gates drives 4 inverters for D_I or D_P; each inverter ready to run 8 inputs. The inputs for I, $\bar{I}$, P, $\bar{P}$ each run 4 inverters, each of which runs 4 inverters, each as I or P ready to run 8 inputs. The central part of the table has the 36 possible values of metric; thus the same metric is given by a #7 D_I associated with I = 1 (likely) and a #8 D_P with P = 1 (very likely); or a #7 D_I I = 1, #1 D_P, P = 0; or a #2D_I, I = 1, #8 D_P, P = 1; or #2 D_I, I = 0, #1 D_P, P = 0. Another four combinations are represented by the commutated case, beginning #8 D_I, I = 1, # 7 D_P, P = 1, for a total of 8 cases. Each of these is a 4 input NAND gate fed by appropriate values of D_I, I, D_P. These 8 gates feed a gate (used as NOR, with inverted inputs) whose output signifies one of the 36 values of metric. Each of these is run through an inverter and feeds the 10 NOR gates gathering terms for the 10 bits of the metric. Eight of the 36 metric gates have only 4 input conditions, as their commutated arrangements are unchanged; thus, for example, #7 D_I, I = 1, #7 D_P, P = 1, located on the diagonal of the imaginary table.

The last stage in the development of the table is shown in Figure 9.00. Here a large saving in the fan-out was achieved by first combining the identical cases of (for example) #7 D_I, I = 1 with #2 D_I, I = 0 to form a ΔM_I level. These eight ΔM_I levels combine with eight ΔM_P levels at the same set of 36 metrics, but with fewer combinations reducing the load on each of the contributors, making the load-sharing inverters unnecessary. The added ΔM_I - ΔM_P level does not add a level of delay, as it replaces the fan out inverters in the timing schedule.

The final bit gates gather the sets of one's and zeroes from the metric levels arranged in order of size on Figure 1007. These values are somewhat arbitrary here and were computed on sets of ΔM_I and ΔM_P as follows:

D_I	I	ΔM_I Number	ΔM_I Value
111	1	111	+0.5
000	0	110	+0.4
etc.		101	+0.1
		100	-0.1
		011	-1
		010	-3
		001	-6
		000	-16

The resultant ΔM values run from +1 to -32 and form a 64-square array whose corners are +1, -15.5, -32, -15.5. Values are duplicated except on the diagonal $\Delta M_I = \Delta M_P$. These values were expressed in binary form to the nearest 1/16 in two's complement form. Since the 36 metrics occur in inverse voltage, the 10 output bit gates are NOR structures. If the metric has a 1 at a particular bit, we tie it in and the output is TRUE. A metric gate with a fan-out of 8 is not called on to drive all 10 outputs as none of the metrics has all one's.

Both the output bit and its inverse are made available by the use of 10 inverters. We can take advantage of this necessity to keep the fan-in within legal limits. The bit gates are limited to a fan-in of 20 inputs. Some of the bit positions have one's represented at more than half of the 36 metrics. For these bit positions we make the meaning of the bit gate a zero and gather inputs from the metrics with zero at that position. The fan-in thus cannot exceed 18 for any table and for this table does not exceed 16. The bit gate output is used instead of the inverter output and vice versa.

The output of the table is used only for addition in the forward direction and only for subtraction in the backward direction. Hence, the control from the Forward-Backward flip-flop is used to switch on the set of bits or the set of inverses. When inverses are used, the function Backward is used as a carry input to the first adder stage to complete the two's complement process of inversion. If the table had been itself arranged in one-complement fashion, so that a negative number to be added would need the carry as well as a positive number to be subtracted, this would have been handled by taking the carry bit for the adder from the sign bit of the table.

The speed of the table: From the time of delivery of D from the memory, the logic settles through 6 gate levels in the final minimized table to the inverse metric bits. The DTL 930 series proved too slow to make a 200 nsec decision available, despite their identification as 30 nsec maximum delay units, as they may take up to 80 nsec in the positive-going direction when lightly loaded. For this reason the greatest part of the gating is specified as the Texas Instrument TTL logic, 7400 series flat pack in commercial temperature range. These are competitive in price, have a fan-out of ten, and maximum delay of 29 nsec. The 7400 gates cannot be connected in the wired-or configuration but the line has an equivalent in the 7450 and 7451 AND-OR-NOT, which similarly allows this function to be performed at the 36 metric gates with a single gate delay. Even before this decision the 10 $\Delta M'$ output bit gates were specified as TTL to ease the interface into the adder which had concurrently developed into TTL logic. The sole remaining DTL logic is at the 10 ΔM gates which require a larger fan-in than is available in the 7400 series. (The Sylvania SUHL TTL has a high fan-in gate but is not compatible with the 7400 and information on this broad line, whose cataloging was recently revamped, was incomplete during the study.) The DTL gate, specified on the drawing with the T.I. numbers 15830 and 15833 for uniformity, is the commercial range flat pack. External load resistors are added to bring the worst delay to 35 nsec. The net delay is 180 nsec to $\Delta M'$ bits from arrival of D from memory, or 29 nsec from Backward-Forward decision.

Adding Δ

When Δ is added instead of a metric, it always happens on a backward move, when a trial subtraction of the metric would have given a negative M. The trial subtraction is of a metric originally added on a forward move, which must therefore be less than +1. Since Δ will always be specified to be larger than +1, the M from which the trial is made is less than Δ . Hence, the higher binary digits of M are then all zero. The low digits of Δ are all zero, and so generate no carries on the addition. With Δ a power of 2, it would be a simple approach to add Δ by forcing the appropriate bit of M to a 1, bypassing the adder and avoiding having to fit this process into the same inputs as the table usually occupies. The danger of such fitting would come in a design involving another level of gating, with attendant delays for the normal table operation. It can be seen from the design history of the table that such trivia as gating of alternatives or coming out with the correct polarity are really major elements of the design. The operational problem in forcing only the Δ bit to one would come on backing down to the beginning of a block. Here one may have to back down to this point more than once to clear a deep valley in the M sequence. On the subsequent additions one is dealing with a remainder larger than Δ . The forcing approach would then require that an artificial region behind the first bit of the block be entered, subtracting out Δ instead of any metric on back moves, forcing Δ on the turn-around, and adding Δ as an artificial metric on the forward moves. This artificial metric, however, is what we had been trying to avoid. The solution was offered by the AND-OR-NOT structure of the 7400 logic which allows us to replace the output bits of the metric table with the value of Δ without incurring another gate delay. This function is incorporated into the output $\Delta M'$ gates in Figure 9.00, with all zeroes except a force to 1 on the DTL gate of the 9th bit, for a Δ value of +16. An otherwise unnecessary gate is shown at the ninth bit, with a dotted connection, in case other values of Δ are required. A value of +8 would be handled similarly at the 8th bit; smaller values would be forced at the TTL metric inverter as these bits gathered one's rather than zeroes from the 36 metric levels. The addition of Δ then would proceed in normal fashion from the table outputs. Time from Add input: 93 nsec. The component count is 15 7400, 16 7420, 10 7450, 9 7451, 5 15830, 14 15833 and 5 7460, for 75 chips.

The Adder

Binary numbers may be added slowly by running their bits sequentially in corresponding pairs through a one-stage full adder, retaining any carries until the next pair. The fastest rate available per bit is the settling time of the carry which is conventionally 3 gate levels. Any attempt to speed the process up by using as many adder stages as bits and hitting them in parallel is completely defeated by the necessity for letting some carries ripple down the adder.[1]

[1]If a variable time is permissible, one may build an end-of-ripple indicator to allow operation to continue whenever the addition is complete. Over the long run one averages a ripple time of $\sqrt{N}$ stages or about 4 adder stages for this problem, about 348 nsec TTL or perhaps 660 nsec DTL. Not only is this average too large but the variable addition time was felt to be inappropriate for the rest of the circuitry being planned.

The literature is full of solutions to this classic problem, going back to Babbage. Only two seemed capable of 200 nsec operation with standard speed I. C. modules and construction:

a) A two-pulse adder using redundant multi-valued notation which would need translation to single-valued form for any reasonable table look-up, the translation taking the form of a conventional adder. Because of the frequent translations, this would be useless.

b) The use of special high-speed circuitry to bypass the carry within the adder stage. We lacked this flexibility, using available forms.

The problem is eased by the fact that, from the algorithm on Figure 2.00, only the sign of the answer need be known before the decision is made to re-access the memory. If the sign is known, the addition and subsequent entry of the answer A as the permanent M can be carried out while the memory is getting access to the next node. Hence, we designed an anticipation network capable of forecasting the sign of A in 87 nsec after a metric is presented; the adder would use a system of clustering stages by 5's, anticipating the carry of a cluster and propagating the carry down the cluster. Approximate time to the sum A was 435 nsec.

At this point in the study, this work was obsoleted by Texas Instruments' announcement of a set of fast adder chips, using the method b) above, all performed within the chip for several stages. The 4-stage 7483 was available only in the DIP package but the 2-stage 7482 is available in flat pack, and yields the second stage carry faster (from an input carry) than a single bit delay. This was worked into the form shown on Figure 8.00, where time to the last bit is 204 nsec and the carry is speeded up slightly by the gating network to be available at 193 nsec. The form of this gating network may be followed from the fact that the ΔM input has a limited range so that its high order bits are simply the same as[1] its sign bit. The first stage and carry input of the 7482,

[1]This "same" convention was chosen so that the negative sign for the table would be a 1, available for use at the first carry input instead of BACKWARD if a one's complement table was decided on.

running the fast carry circuitry, represent four normal loads, which requires the double inverter on the ΔM_S input. The output A bits are fed to the M latch circuits; if it is decided to set M = A, the M clock is activated.

Parity Generation

The 15 parity terms are shown in Figure 4.00. A parity of 1 indicates an odd number of 1 terms. One could have a number of gates of 15 inputs, showing the one way of having 15 1's, the 210 ways of 13 1's and 2 0's, etc., all OR'd together, but this is not only a huge number of gates but requires a fan-in of 15 at the first level and a monstrous fan-in at the OR. On subdividing this structure, we find that gathering terms by 4's still requires too big a fan-in at the OR. Gathering by 3's gives good fan-in and acceptable loading on the source.

The gating used first gathers up the parity in 3 terms. There are 5 such parities developed. Using the TTL AND-OR-NOT form, they take only 1 gate delay, 29 nsec. We gather up an even number of ones for this form, since the gate then inverts the answer. Using the parity and $\overline{\text{parity}}$ from these first levels, we repeat the process at 2 second levels where we develop the parity of the first three parities and the parity of the last 2 parities. In order to avoid the delay of an inverter to get $\overline{\text{parity}}$, we duplicate the parity structure for inverse inputs to get even parity, which is $\overline{\text{parity}}$. The second total delay is 87 nsec to this point; the restrictions on the design having been to minimize the number of successive levels and the number of inputs per gate, rather than the more usual total number of inputs or total number of gates.

For loops in which the first hypothesis for H is unsuccessful and we must reverse it, we wish to avoid the process of setting the reversal, having it take effect at H, and then letting the result ripple down the 87 nsec generator. The resultant P is always the reverse of the initial P, so that we can save time by applying the reversal to P itself. This is shown (in Figure 14.00) by making reversal one of the inputs to the third level of the generator.

Speed of the Algorithm

The consideration of the medium-speed decoder was at this point divided into two subsections; first, a simpler system using the natural access time of the core memory, and leading to a variable node time averaging slightly in excess of 1 microsecond; and second, a fast access system holding all node times to exactly 1 microsecond. Timing of the natural system is examined first.

The memory was assumed to be one which has a basic cycle of 900 nsec; 1000 nsec cycling in the READ-MODIFY-WRITE mode; and 350 nsec access time (one of the advantages of the fast access system is that the natural access time ceases to be an important parameter). The model here is the Electronic Memories Nanomemory 900.

We assume initially a basic clock rate of 5 mc. Not every 200 nsec period has a clock pulse output; for example, the table feeds into the adder without the necessity of clocking. At 350 nsec, D is available from the memory for application to the table, asynchronously. I and P are already present there; note that the speed of P generator is not critical in this application as long as it is less than the access time. At 530 nsec ΔM is ready. The sign of A, A_S, is ready at 723 nsec. At 752 nsec A_S has been gated with other appropriate factors and is ready for a decision.

If the first try, H = 0, is successful, the 4th clock pulse at 800 nsec interrogates the decision, which is for a forward access and right shift. This is a true pulse. The pulse will pass the gate and last from 829 to 849 nsec; its trailing edge shifts the H register which is settled down by 899 nsec. At the same point, the memory address counter has settled to its new value. M has been set to equal the trial metric A, and is also ready by 929 nsec. The new value of H is automatically zero, by the shifting process. At 1000 nsec the new memory access is initiated again. Meanwhile, the D values were rewritten into the memory and the oldest value of H written in by a gated pulse from 429 to 592 nsec as required by the core memory. (The circuit needn't wait for the decision on H, as it isn't the current value that is entered into memory but the oldest bit from the last forward node operation.)

If the try H = 0 is unsuccessful, the 4th clock at 800 will not shift but will initiate the reverse I and P examination. The new H will be ready at 858, A_S at 1231, gated at 1260, interrogated by the #7 pulse at 1400 and the memory addressed again by the #8 pulse at 1600. Since the H = 0 case cycled in 1000 nsec, the average node time on a forward step is 1.3 microsecond.

To improve this time, we may vary the clock frequency to take advantage of the long pauses with data ready before the clock strikes. We need not restrict ourselves to addressing in 1 microsecond for H = 0 if thereby we can lower the average time by speeding up the H = 1 cycle. We call operation at 5 mcps mode 2: interrogating 0 at #4, addressing 0 at #5, interrogating 1 at #7 and addressing 1 at #8. Results of some other modes are given below. These are for the fastest clock rate possible for the particular mode; a slower rate in each mode will merely increase the cycle time in direct proportion to the clock period.

Mode	Period	Rate	Try 0	Address 0	As Gated	Try 1	Address 1	Average Cycle
1	251	3.98 Mc	#3 753	#4 1004	1213	#5 1255	#6 1506	1255
2	200	5.00	#4 800	#5 1000	1260	#7 1400	#8 1600	1300
3	167	5.98	#5 835	#6 1002	1295	#8 1336	#9 1503	1253
4	154	6.49	#5 770	#7 1078	1230	#8 1232	#9 1386	1232
5	151	6.62	#5 755	#7 1057	1215	#9 1359	#10 1510	1284
6	125.5	7.97	#6 753	#8 1004	1213	#10 1255	#11 1380.5	1143

Other modes are possible, and this is not a complete search. A period of 125 nsec is perhaps hazardously short, so that the mode 4 is perhaps the optimum. The time is sensitive to exact memory performance; note that if the memory could be recycled in 924 nsec, the clock rate of mode 4 could be used in mode 3, with an average forward cycle time of 1155 nsec.

Cycle times for Backward moves will average slightly longer, owing to the moves in which we move back on an old zero, fail to get A_S negative, must set M = A and then try 1, fail again and continue backward. Setting of M = A runs concurrently with trial of 1. These figures have not been calculated but will run perhaps 100 nsec longer. The average time over-all will be close to the averages of backward cycle times and forward cycle times (for the high error rates under discussion) since the node circuit will spend most if its time in long forward or long backward searches, with few reversals of direction and only slight forward bias (from simulation results).

Medium-Speed, Fast-Access System

We can hold the cycle time of the medium-speed decoder to exactly one microsecond by doing away with the effect of the long access time to the memory. Figure 12.00 shows the required arrangement for channel bit access. An immediate access section, i.e., about 70 nsec, is used as a "hot memory." This consists of three-stage shift registers, six abreast. The central stages are the current D. At address time the hot registers shift so that D_{n+1} shifts to center, D_n to the right stages, and D_{n-1} is wiped. At access time the memory puts D_{n+2} into the left stages. At write time the memory accepts D_n from the right stages, into the cores just vacated by D_{n+2}. The hot memory is thus in series with the main memory ring. The process makes it unusually clear that the core address number is simply an address, and not the identification of a branch. As the search continues forward, the looped connection to the hot memory slides around the ring, with the bulge of three nodes maintained up to date within it, giving Figure 12.00 the appearance of an orange being swallowed by an ostrich.

The parity generator timing now becomes a critical item. A small hot memory is constructed for this in the form of a parity generator preparing parity for the next move if forward and one preparing what it would be if backward. Parity itself is then picked up in a flip-flop on the memory address time. For backward moves the old H is one of the components. A new Forward-Backward level must be added to the parity gates bringing the time to 116 nsec, but this is now not critical in the 1000 nsec cycle.

With the series hot memory two complete table lookup and adder operations may be fitted into the 1000 nsec cycle. Detailed timing has not been worked out but any small interference in clock spacing can be overcome by the following modification. Break the table at the 36 metric gates. Add a level of gating to provide Normal connections as drawn and Reverse connections from mirror image metric gates. This will slow up the initial trial of H = 0 but greatly speed up the process of then finding the metric for H = 1. The use of separate tables would yield even more time margin but is unnecessary.

Estimate of Material for Medium-Speed Fast-Access Decoder

The following summary shows estimated value of the memory and of the integrated circuits for the decoder. The integrated circuit estimates are known with some precision for the adder, table, and parity gating since these were investigated in detail as items of critical timing. The parity gate of Figure 14 is doubled on the summary to represent the double choice available for the next move in the rapid access scheme. The asterisks for 7451 circuits indicate that spare circuits are available for these functions from 7450's in other sections; no 7451's are actually used. Current Texas Instrument prices are used, but items similar to the 15800 line are widely available. The summary uses price breaks for the quantities indicated for one decoder.

The memory indicated follows the size of a standard 3C memory of similar capacity, but the price indicated is intended to represent a general class of 1 microsecond memories and is not based on a formal quote from any manufacturer (we are awaiting these).

The cost of packaging the integrated circuits may be estimated based on the common arrangement of cards plugged into files with back-plane wiring. This arrangement is especially useful for one-off manufacture. A survey of a variety of such lines shows a card cost of about four times the cost of the integrated circuits, for a total of about $5,000. Special card design costs will amount to about $500.

The total number of cards will run to perhaps 75, which will fit into a 3C Type BL330 Cabinet and require a power supply similar to the 3C PB331, mounted within the cabinet.

The Memory Cabinet is about 12 1/4" x 5 1/8" x 8 1/2" and occupies one-half of a relay rack space 5 1/8" high. The Main Circuit Cabinet is about 12 1/4" x 5 1/8" x 17" and occupies a full relay rack space 5 1/8" high.

TOTALS:	Logic Cards	$5,000
	Card Layout Costs	500
	Memory	3,000
	Memory Power	200
	Main Power	365
	Main Cabinet	345
	Clock	250
	Total Material	$9,660

Summary--Medium-Speed Sequential Decoder

MEMORY

1 256 x 8, 1 us Core Memory in 3C BM335 Case;	$3,000
1 Power Supply PB330, 3C	200

CENTRAL PROCESSOR

	SN7400	7420	7450	7451	7460	7430	7482	15830	15833	7472
Look-up Table	15	16	10	(9)*	5	0	0	5	14	0
M-Store	9	0	0	0	0	0	0	0	0	0
Adder	15	0	0	0	0	2	7	0	0	0
BRANCH TRACER										
H-Store	20	0	0	0	0	0	0	0	0	26
Parity Generator	2	0	30	(2)*	58	0	0	0	0	1
Rapid Access Memory	63	0	0	0	0	0	0	0	0	21
Clock Drive	10	0	0	0	0	0	0	0	0	0
Control Logic	5	0	0	0	0	0	0	0	0	4
MEMORY ADDRESS										
Tortoise	0	0	0	0	0	0	0	0	0	9
Hare	9	0	0	0	0	0	0	0	0	9
Detect & Drop	8	8	0	0	0	0	0	0	0	2
BLOCK SYNC										
Block Address	0	0	0	0	0	0	0	0	0	9
Block Key	5	0	0	0	0	0	0	0	0	10
FORCE										
7th Bit Force	2	2	4	0	0	0	0	0	0	4
Force Block	4	0	0	0	0	0	0	0	0	0
TOTALS	167	26	44	0	63	2	7	5	14	95
							Total Chips:	448		
UNIT PRICE	2.50	2.90	3.20	3.65	2.30	3.65	8.60	3.90	3.45	3.50
TOTAL PRICE	417.50	75.40	140.80	0	144.90	7.30	60.20	19.50	48.30	332.50
							Total Chips:	$1,246.40		

The Ultimate Speed Machine

A speed gain of about 3 or 4 is obtainable in the digital processor by use of ECL logic (RCA, Motorola, Fairchild); perhaps a little less by use of extra-high speed TTL (Texas Instruments, Sylvania SUHL II). The exact gain in speed becomes harder to predict and dependent on the physical packaging. This high-speed logic may be used with standard core memories of perhaps 650 nanosecond cycle time (750 nanosecond READ-MODIFY-WRITE) and maintain the speed advantage by the use of a block hot memory. Unlike the series hot memory described above, this would pull out several nodes' worth of data at a time. For example, it might practically pull 7 bauds of 6 bits into a hot scratchpad divided up 7 bauds wide each side of the current baud. The core memory would thus be organized with a 42 bit word structure. The node circuit could then jitter about within the 14 baud region at high speed. On reading one end, the other half would be dumped back into the main memory, the 7-baud half retained as back guard for the current node, and the empty half filled with the next 7 bauds x 6 bits from main memory. The net gain in speed from bursts of 7 baud searches has not yet been simulated but should be appreciable.

For high speeds, one may well wish to alter the nature of the main memory, as well as its organization. Consider the Raytheon Biax memory which has Non-Destructive Read-Out at a 300 nsec cycle time. This could be used for the node speed access which needs NDRO operation or must simulate it by rewriting. These memories are organized in linear select mode rather than coincident current; this allows the economic arrangement into the large word size with the small number of words required for the block hot memory scheme. At 300 nsec node cycle, it might prove worthwhile even without block hot memory in an upgraded medium-speed decoder. The present difficulty with this memory is that the write mechanism lasts 10 microseconds. The write mechanism would only be used by the channel rate circuits; but even for these at a one-megacycle channel rate, we must be able to write a 7 baud word (7 microseconds real time) in a very small fraction of 7 microseconds to allow interlacing with the node speed accesses. The write speed restriction is not inherent in the memory but development would be needed to obtain the needed write time of perhaps 700 nsec.

Another future possibility lies in the plated wire memories of Librascope - (General Precision). These are Non-Destructive Read-Out with 100 nsec cycle, are most naturally organized in linear select for large words (although at this speed a block hot memory is probably unnecessary), and can write extremely fast. In addition, the planes can be woven into permanent read-only memories for use as the look-up table, or may be most conveniently built for that service as read-only, electrically alterable. In the latter form the write circuits are commutated to follow, for example, a punched tape input to alter the metric table to suit altered channel conditions. The restriction on these memories is that the manufacturer offers only the memory planes. Complete memory systems would have to be developed, a condition we decided was unsuited to medium-speed decoders but tolerable in the ultimate speed machine. In addition a large set-up charge is made for the automatic loom if non-standard planes are desired.

The final element examined was the fast parity generator. Buffering the parity gates into a flip-flop provided quick access for the medium-speed decoder, but the size of the gate structure would begin to restrict the clocking rate in a high-speed machine. For this reason a generator was developed with only a single operation between clock pulses.

The basic scheme is shown in Figure 5.00. The generator is a 14-stage shift register, coupled through exclusive-or circuits. On each shift, another of the 15 terms of parity is added in. If P_{25} is in question, the terms range from I_1 to I_{25}, as seen in Figure 4.00. When I_1 is located in the 14th stage of the I register, it is impressed on the A stage of the generator. At the clock it enters A (and the 15th stage of I register). I_2, the next term, is now in the 14th stage, so that the modulo-2 sum of I_2 and I_1 (in A) is impressed on B, and entered on the next clock pulse. Finally, all the terms but I_{25} are present in the left stage of the generator while I_{25} sits in the left stage of the I register. I_{25} itself is not needed yet as a component of P_{25} as it is always 0 initially and in fact is not available for that service.

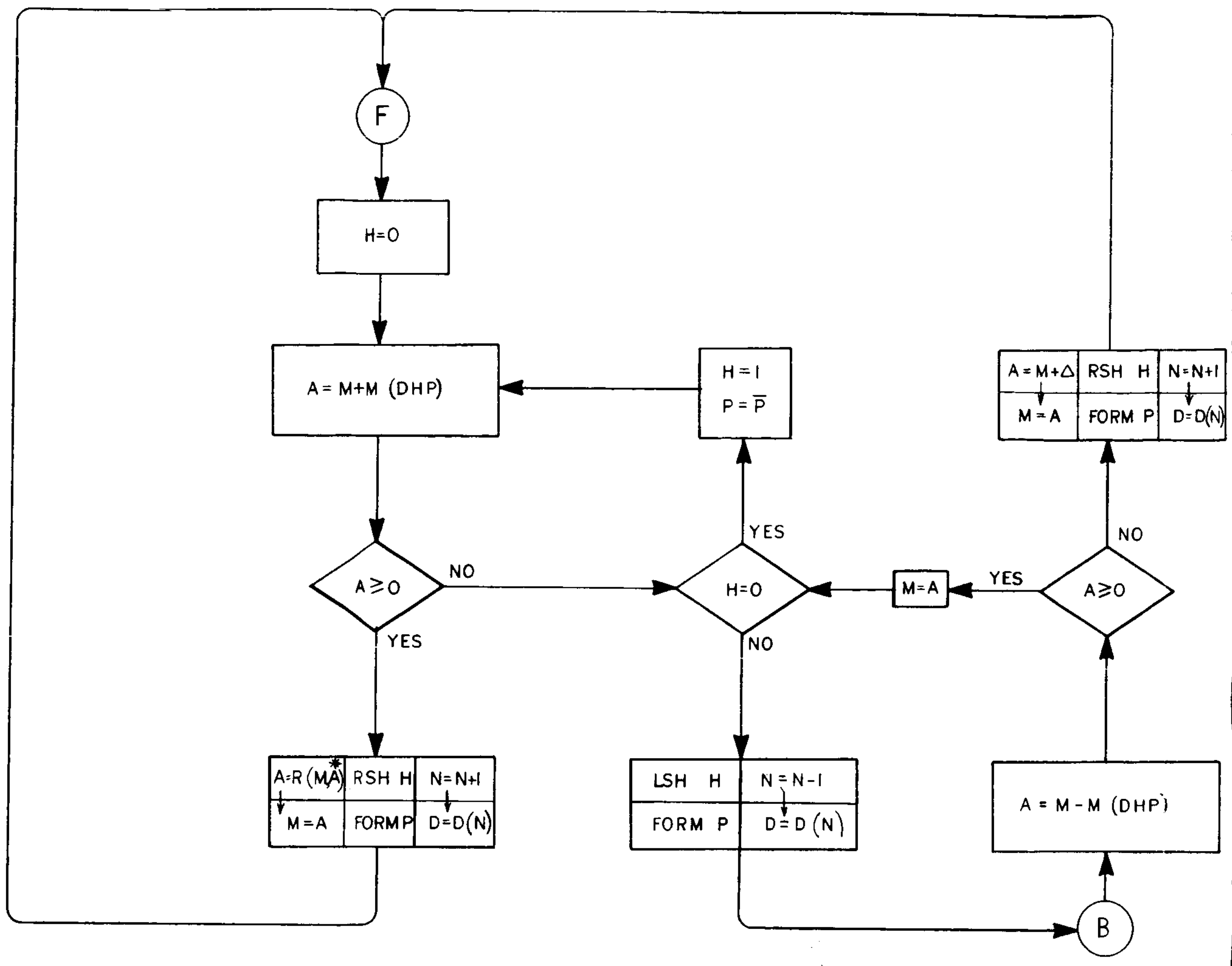
This generator is adequate for forward-shifting I registers. However, the I register may be called on to shift backward, and even to do so after all the terms for P_{25} have vanished into the main memory. To allow the process to work either way, more stages are added in which first the "final" value of I_{25} is added to P_{25} and then at each step another component is stripped out as the partial P_{25} passes down the generator. This is shown in Figure 6.00. On shifting to backward motion at any time of this process, the stripped components are reinserted from their new locations, as shown in Figure 7.00. This shows the same stages as the forward drawings, but with a second set of I register inputs. In backward travel, P_{25} must reflect the actual condition of I_{25} since the algorithm decides whether to alter it. Hence, for forward travel we read I_{25} and P_{25} at a and M, but for backward travel at b and N, the latter containing the full value of P_{25}. That is, it contains the value of I_{25} at the time it is read. After passing backwards, the components are stripped off again in reverse order by double addition $(A + A = 0)$.

The table on the next page shows a forward-backward-forward sequence with this shifting pick-off point. It also shows how the original value of I_{25}, called I_{25-a}, is caught on the backward move and altered to I_{25-b} while the effect of I_{25-a} is removed from both P_{25} and P_{26} (in which I_{25} enters as I_{24} does in P_{25}). The asterisk shows the stage being used for read out.

The complete schematic for the generator is shown in the two sheets of Figure 3.00. The only points of design interest here are that the I register is coupled by NOR circuits rather than the expected NANDS, so that Backward and Forward feeds to the parity register may be kept separate. When the Backward drive is in use, both FI and $\overline{\text{FI}}$ drives to the registers go false; thus, at the parity stages joint exclusive-or input, the forward section of the parity coupling is simply out of action. Because of the NOR action as an inverted NAND, the function Backward is supplied to the actual forward coupling and vice versa.

A more subtle point can be followed on Figure 3.00 (and 7.00). The Forward-a component applied to parity stage N is the exclusive-or version so that the final value of I_{25} can enter them into P_{26}; and into parity stage 0 to enter into P_{25}. However, the backward-b component applied to parity stages N and M is the plain b, not the exclusive-or version, since it is the old value of I_{25} that we wish to remove from P_{25} and P_{26} on a back move. The arrows from stages a and b in Figures 6.00 and 7.00 show this distinction.

Memory Access	Shift Pulse	Last Direction	Memory Address	I Register Content a ↓ b ↓ c ↓	P Register Contents P ↓	O ↓	N ↓	M ↓ P_{25}	L ↓ P_{26}
X	•⟶	F	N-1	24* 23 22 21---	----	----	-*--	22..,2,1	...3,2
X	•⟶	F	N	25* 24 23 22 21 --	----	----	24,22*..21	23..,3,2	4,3
X	•⟶	F	N+1	26* 25a 24 23 22..21-	----	25a,24,22,21	25a,23..3,2	24..,4,3	5,4
X	•⟶	F	N+2	27* 26 25a 24 23 22..21	25a 24 22 2	26 25a 23.. 3, 2	26 24*..4,3	25a.,5,4	6,5
X	None Swap	B	N+1	27 26* 25a 24 23 22..21 -	25a 24 22 2	26 25a* 23 3, 2	26 24...4,3	25a..5,4	6,5
X	⟵•	B	N	26 25a 24 23 22..21 --	----	25a 24* 22 2,1	25a 23..3,2	24...4,3	5,4
None Swap	⟵•	B	N	25b 24 23 22 ..21 --	----	----	24 22...2,1	23...3,2	4,3
X	•⟶	F	N+1	26* 25b 24 23 22 21 -	----	25b 24 22 ...21	25b 23*.3,2	24...4,3	5,4
X	•⟶	F	N+2	27* 26 25b 24 23 22 ..21	25b 24 22 ...2	26 25b 23 ..3,2	26 24*..4,3	25b..4,3	6,5



* R (μ,A) = A

R (M,A) = [A]

WHERE M < Δ ≤ μ

#/

FOLDOUT FRAME 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

2

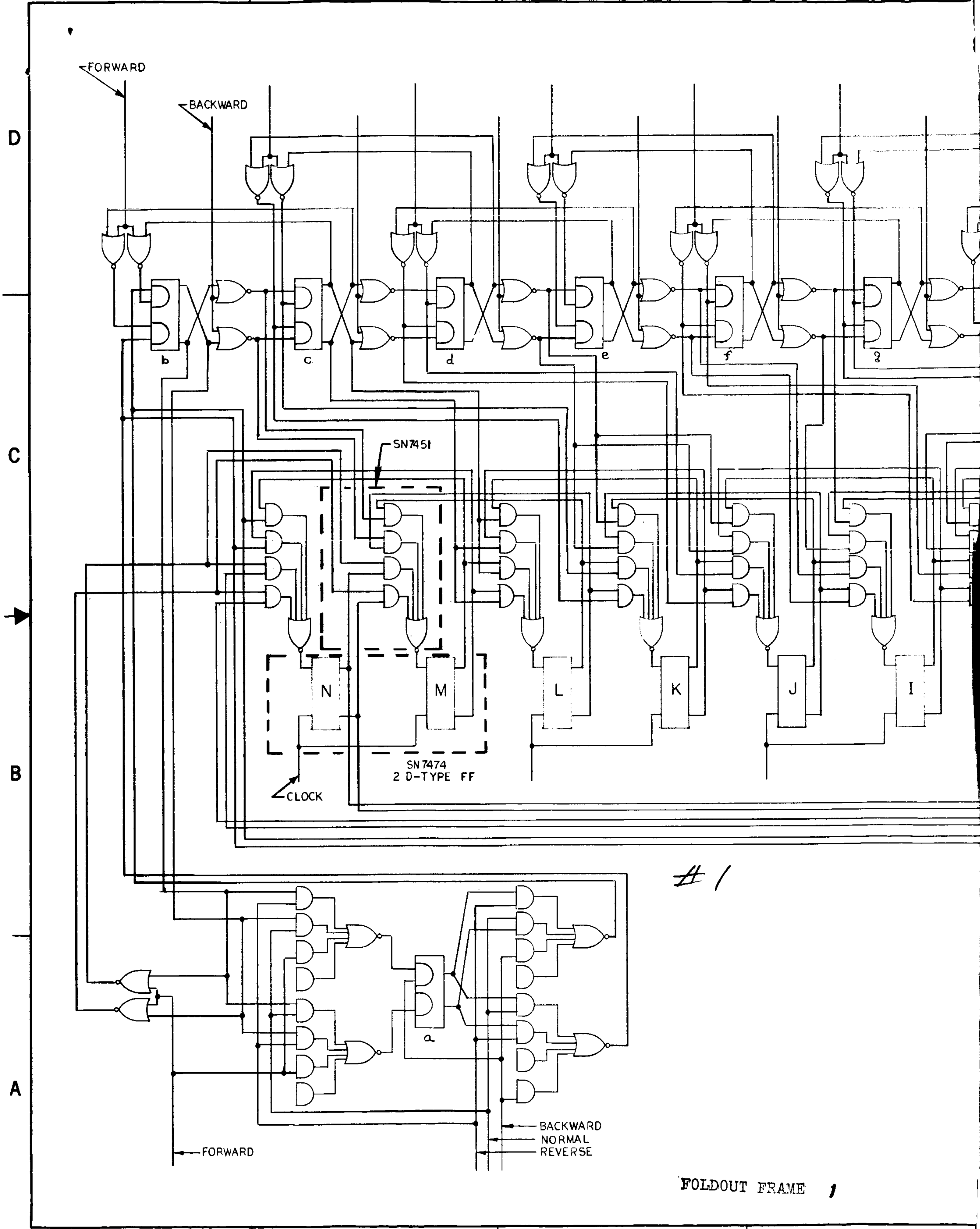
FOLDOUT FRAME 2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

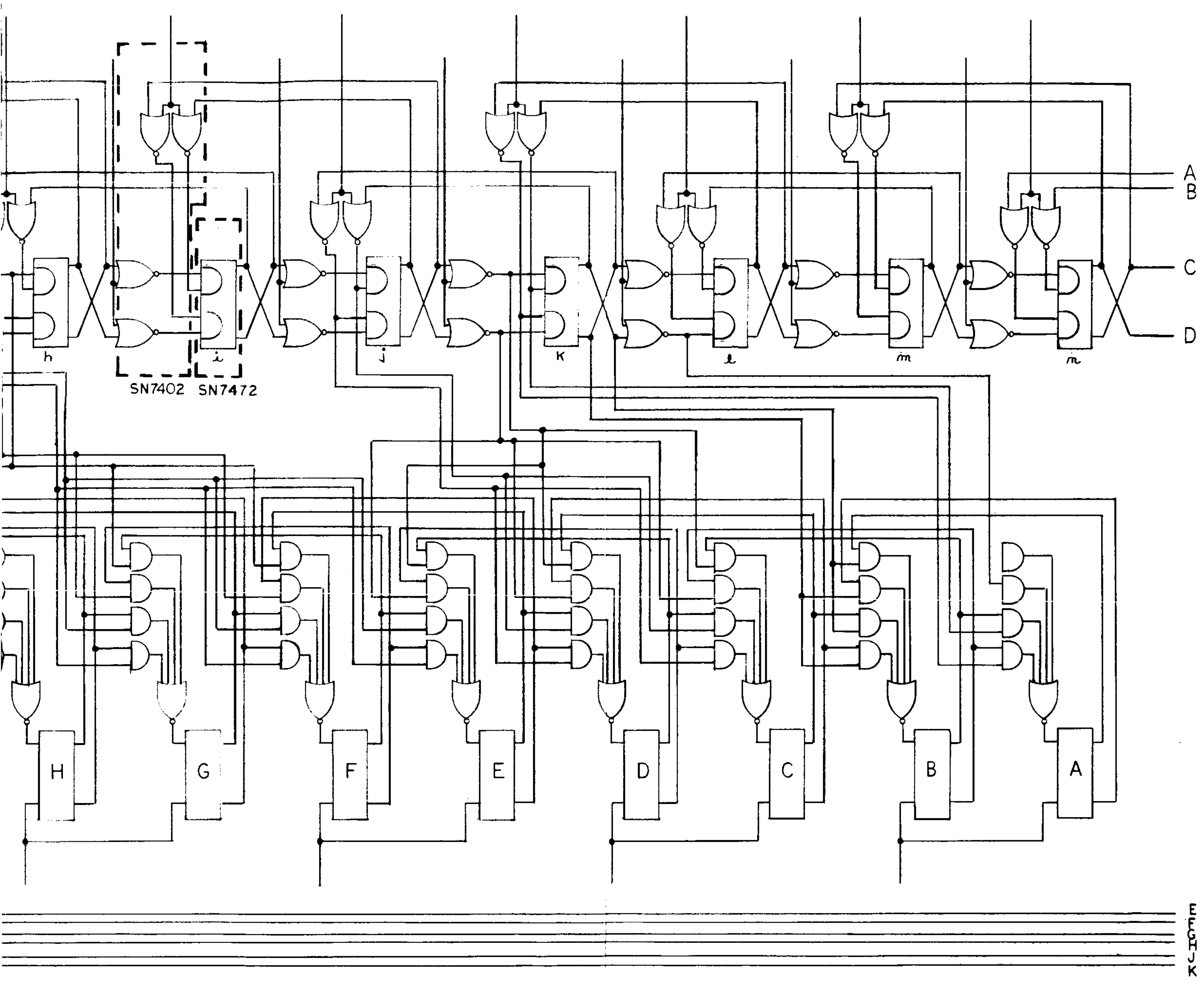
KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR A.W. KELLAWAY DATE 4-26-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± ; SURFACES √ MICRO INCHES	CHK [illegible] 4-30-67	SEQUENTIAL DECODER COMPACT ALGORITHM
		APPD [illegible] 4-30-67	
		APPD [illegible] 4-30-67	
FINISH	MATERIAL		SIZE D — CODE IDENT 25420 — F028 -2.00
NEXT ASSEMBLY — MODEL			SCALE NONE — SHEET 1 OF 1



FOLDOUT FRAME 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



#2

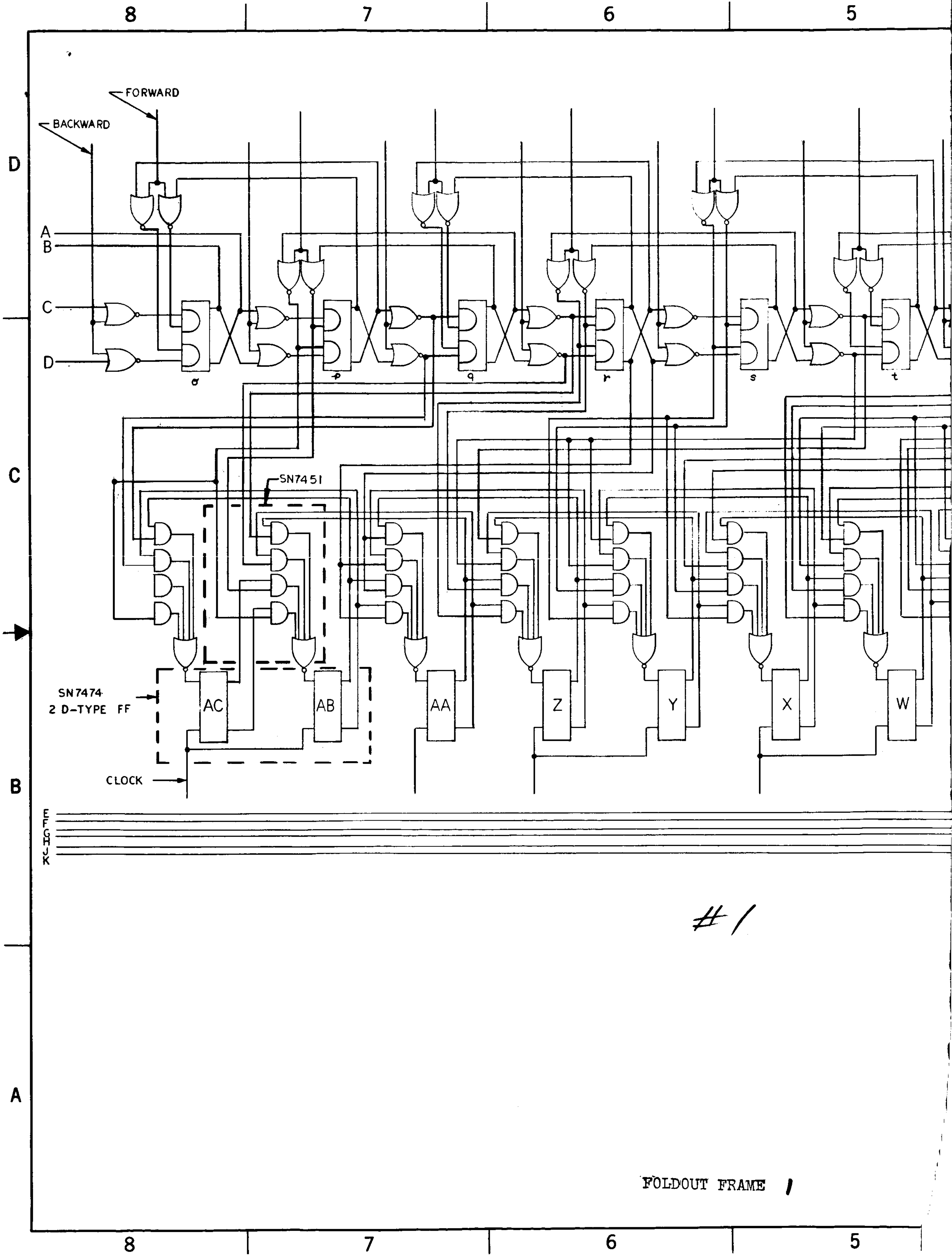
FOLDOUT FRAME 2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °-VENDOR ITEM-

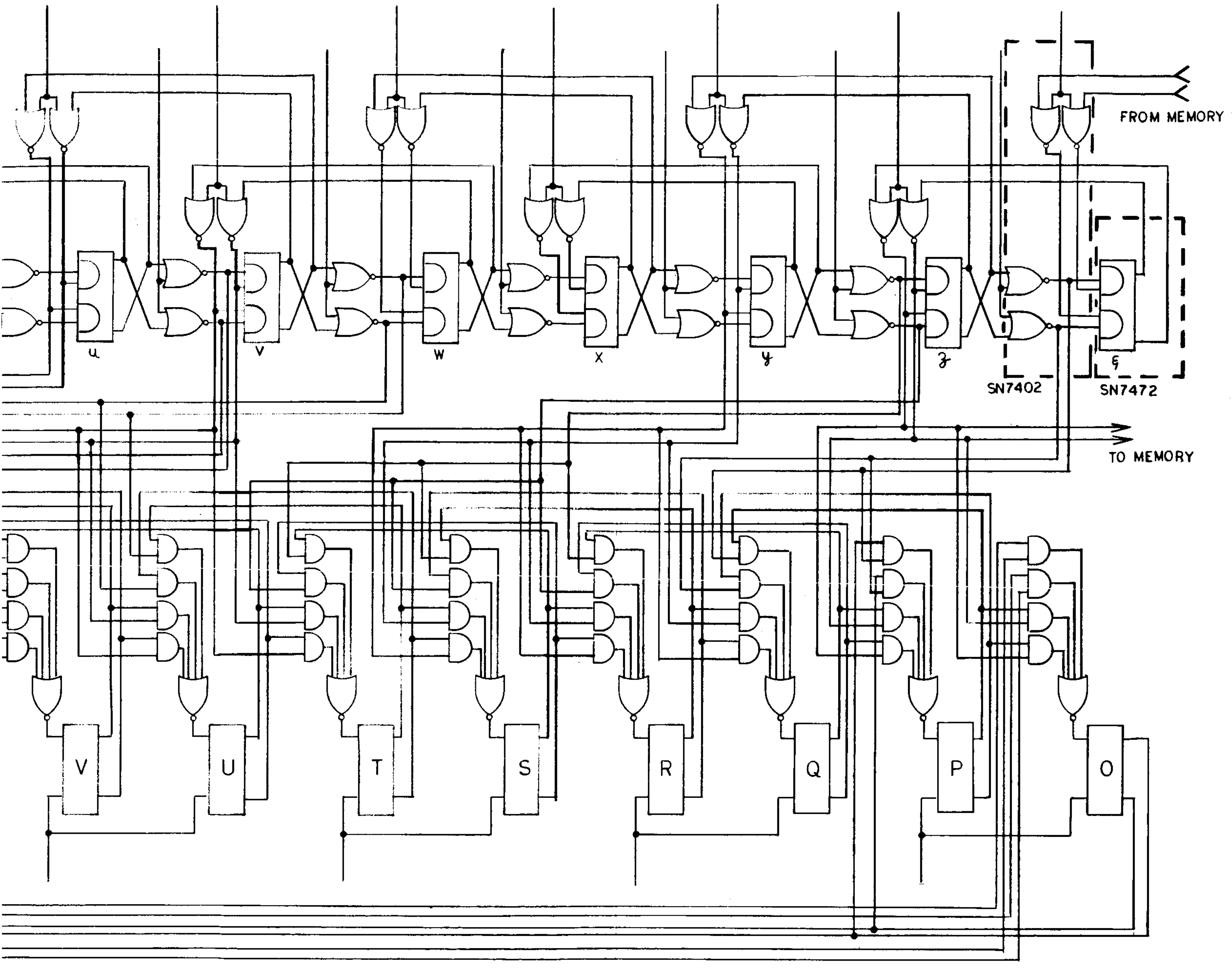
APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S TRUE DATE 4-26-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± SURFACES √ MICRO INCHES	CHK APB 4-30-67	SEQUENTIAL DECODER TWO WAY ENCODER
		APPD AeBeme 4-30-67	
		APPD APB 4-30-67	
FINISH	MATERIAL		SIZE D CODE IDENT 25420 F028.03.00
NEXT ASSEMBLY / MODEL			SCALE — SHEET 1 OF 2



#/

FOLDOUT FRAME 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



#2

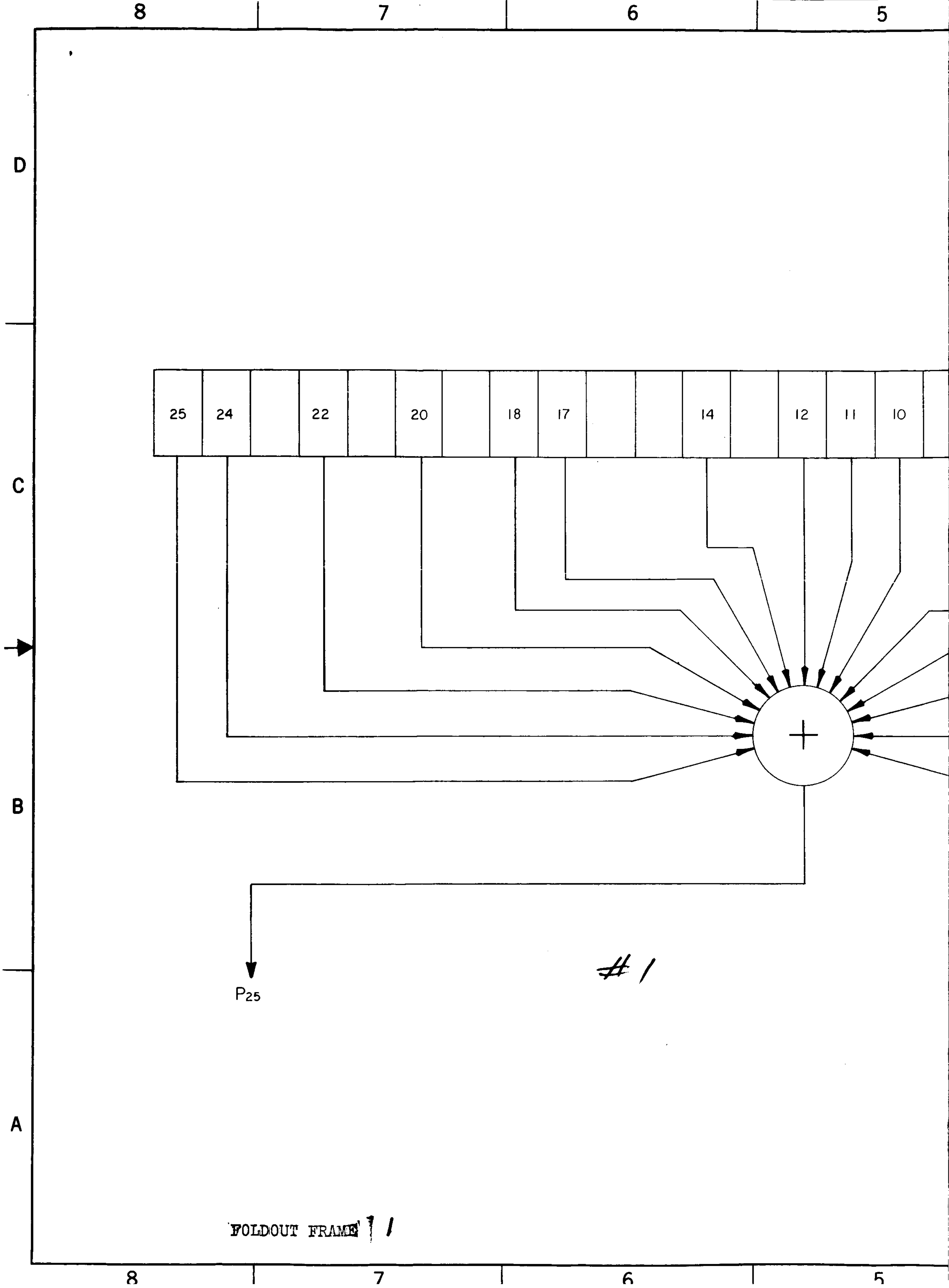
FOLDOUT FRAME 2

					KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °-VENDOR ITEM-

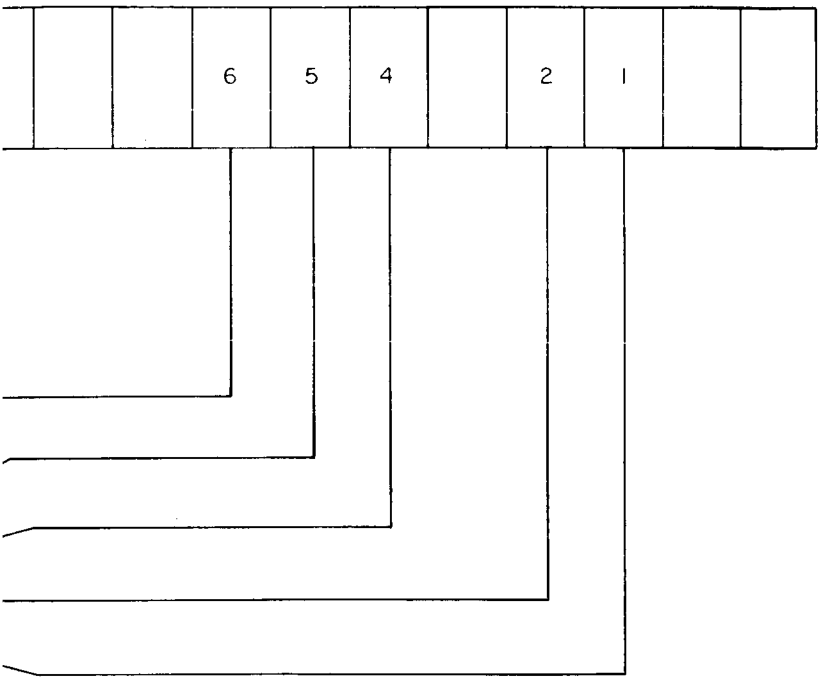
APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S TRUE DATE 4-27-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ±	CHK APB 4-30-67	SEQUENTIAL DECODER TWO WAY ENCODER
	SURFACES √ MICRO INCHES	APPD [illegible] 4-30-67	
		APPD APB 4-30-67	
FINISH	MATERIAL		SIZE D CODE IDENT 25420 F028.03.00
NEXT ASSEMBLY / MODEL			SCALE — SHEET 2 OF 2



#1

FOLDOUT FRAME 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



#2

FOLDOUT FRAME 2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS

FINISH

NEXT ASSEMBLY | MODEL

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES

TOLERANCES
ANGLES ±
DECIMALS
2 PLACE ±
3 PLACE ±

SURFACES ✓ MICRO INCHES

MATERIAL

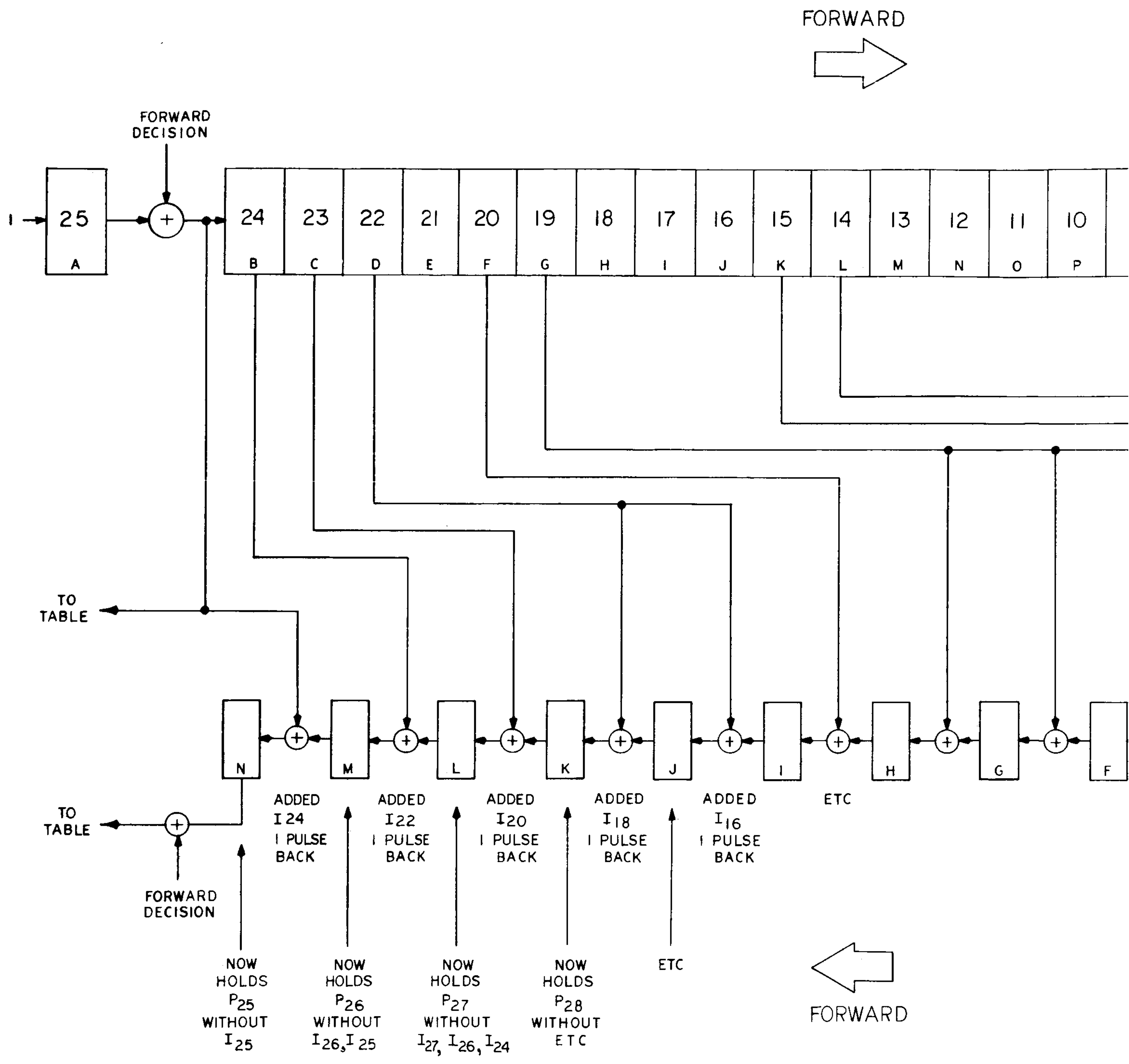
DR	A.W. KELLAWAY	DATE 4-28-67
CHK	[illegible]	4-30-67
APPD	[illegible]	4-30-67
APPD	[illegible]	4-30-67

CODEX CORPORATION
222 ARSENAL STREET
WATERTOWN, MASSACHUSETTS

SEQUENTIAL DECODER
PARITY GENERATOR

SIZE	CODE IDENT	
D	25420	F028-4.00

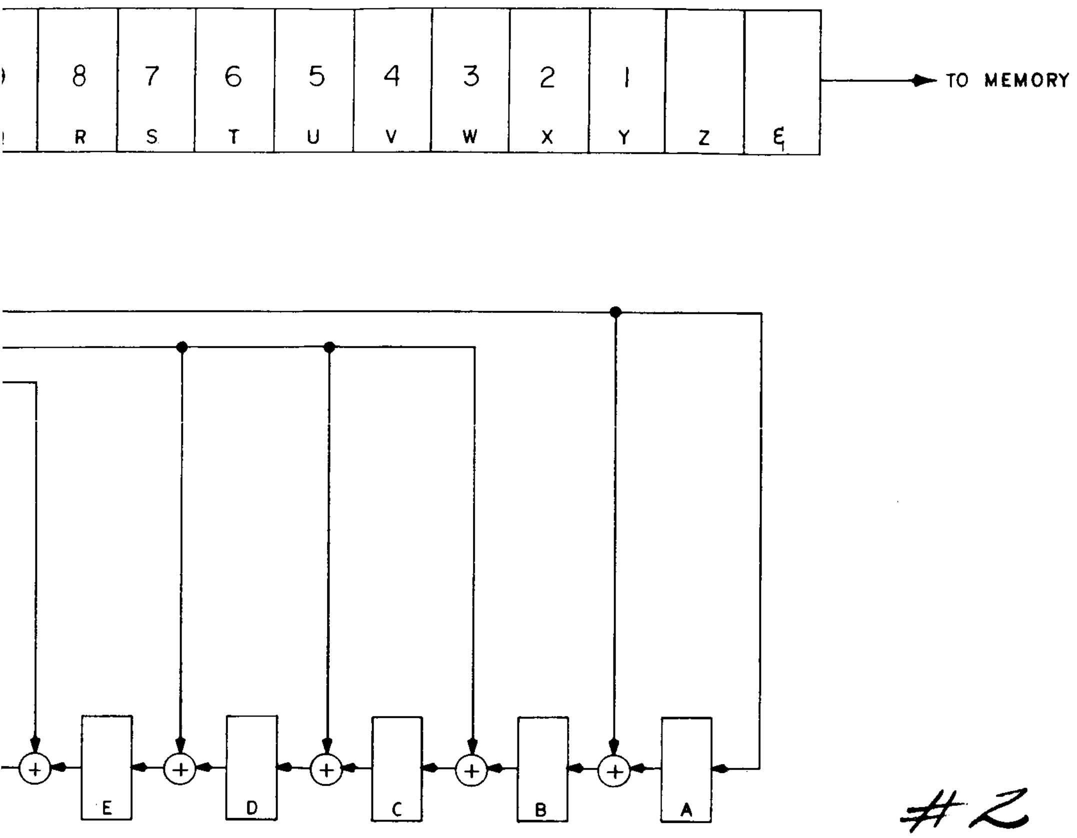
SCALE NONE | SHEET 1 OF 1



#1

FOLDOUT FRAME -1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



#2

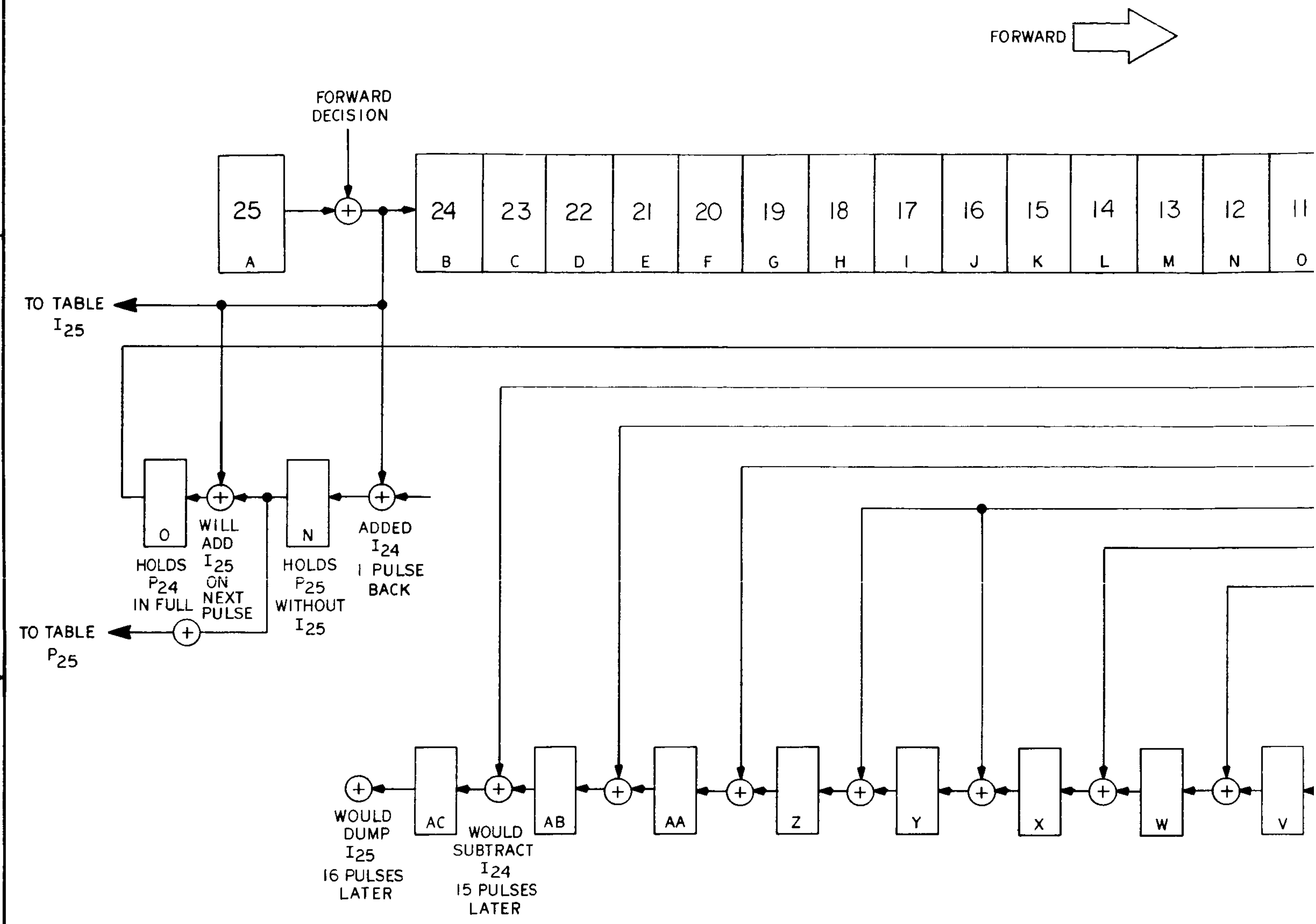
FOLDOUT FRAME - 2

				KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

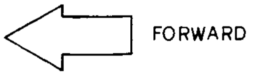
KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR A.W. KELLAWAY DATE 5-1-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES	CHK	
	ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± / SURFACES √ MICRO INCHES	APPD [illegible] 5-2-67	SEQUENTIAL DECODER FORWARD PARITY ACCUMULATOR
		APPD	
FINISH	MATERIAL		SIZE D / CODE IDENT 25420 / F028-5.00
NEXT ASSEMBLY / MODEL			SCALE NONE / SHEET 1 OF 1



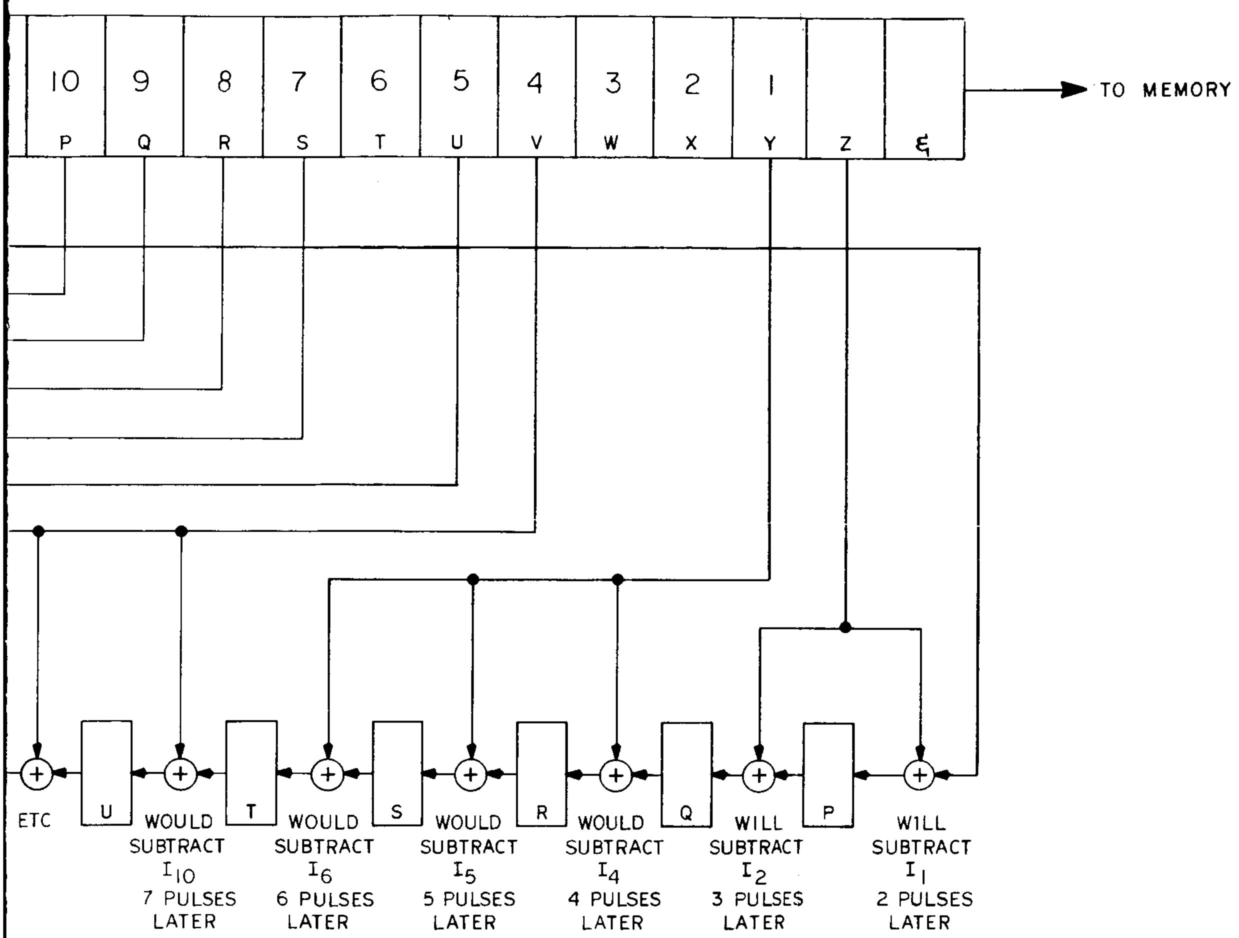
FORWARD
FORWARD DECISION
25
A
24
B
23
C
22
D
21
E
20
F
19
G
18
H
17
I
16
J
15
K
14
L
13
M
12
N
11
O
TO TABLE I_{25}
O
HOLDS P_{24} IN FULL
WILL ADD I_{25} ON NEXT PULSE
N
HOLDS P_{25} WITHOUT I_{25}
ADDED I_{24} 1 PULSE BACK
TO TABLE P_{25}
WOULD DUMP I_{25} 16 PULSES LATER
AC
WOULD SUBTRACT I_{24} 15 PULSES LATER
AB
AA
Z
Y
X
W
V

#1



FOLDOUT FRAME -1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



10
9
8
7
6
5
4
3
2
1
P
Q
R
S
T
U
V
W
X
Y
Z
ε
TO MEMORY
ETC
U
WOULD SUBTRACT I_{10} 7 PULSES LATER
T
WOULD SUBTRACT I_6 6 PULSES LATER
S
WOULD SUBTRACT I_5 5 PULSES LATER
R
WOULD SUBTRACT I_4 4 PULSES LATER
Q
WILL SUBTRACT I_2 3 PULSES LATER
P
WILL SUBTRACT I_1 2 PULSES LATER

#2

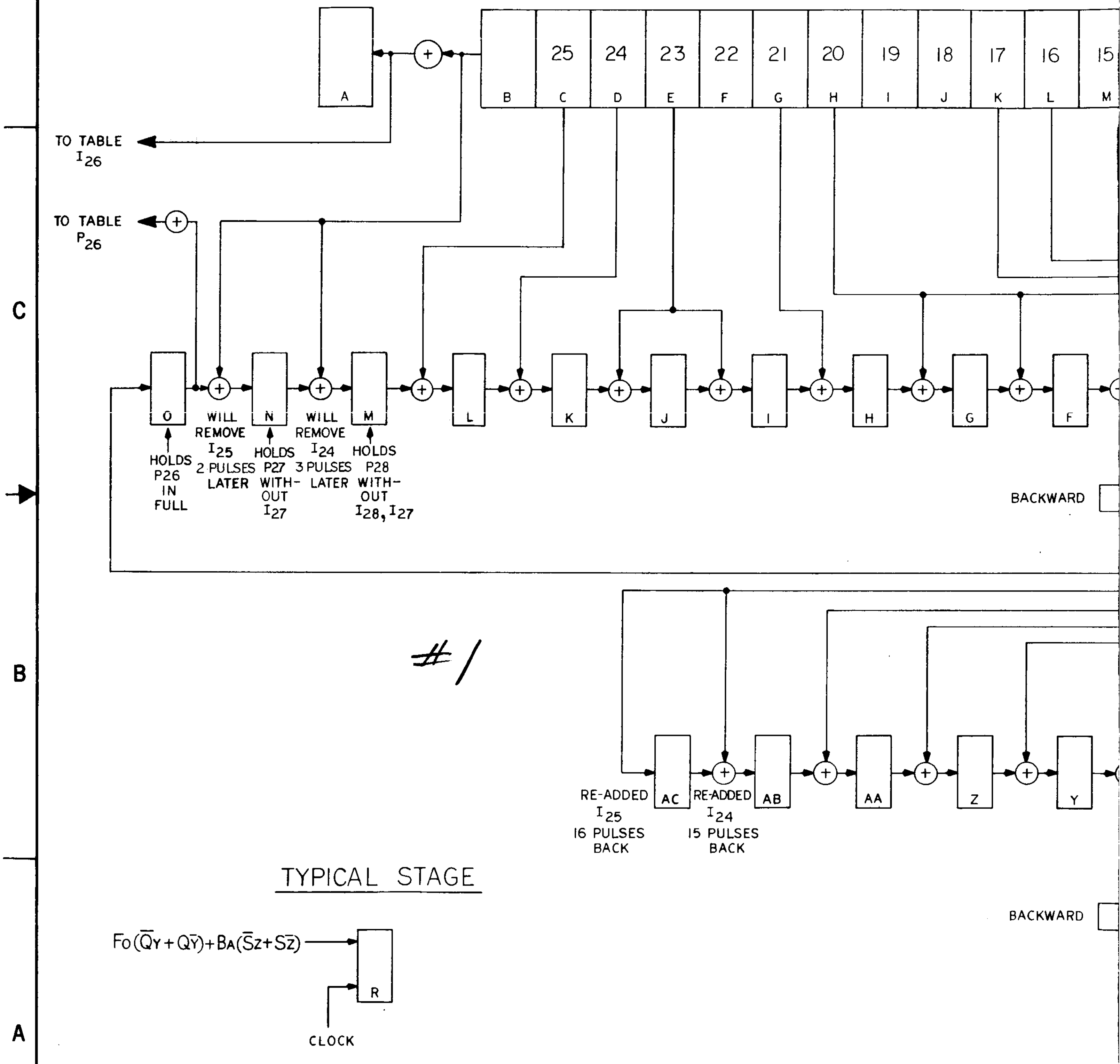
FOLDOUT FRAME -2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR A.W. KELLAWAY DATE 5-2-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ±	SURFACES ✓ MICRO INCHES	CHK
			APPD [illegible] 5-2-67
			APPD
FINISH	MATERIAL		SEQUENTIAL DECODER FORWARD PARITY DIVESTITURE
NEXT ASSEMBLY	MODEL		SIZE D	CODE IDENT 25420	F028-6.00
			SCALE NONE	SHEET 1 OF 1



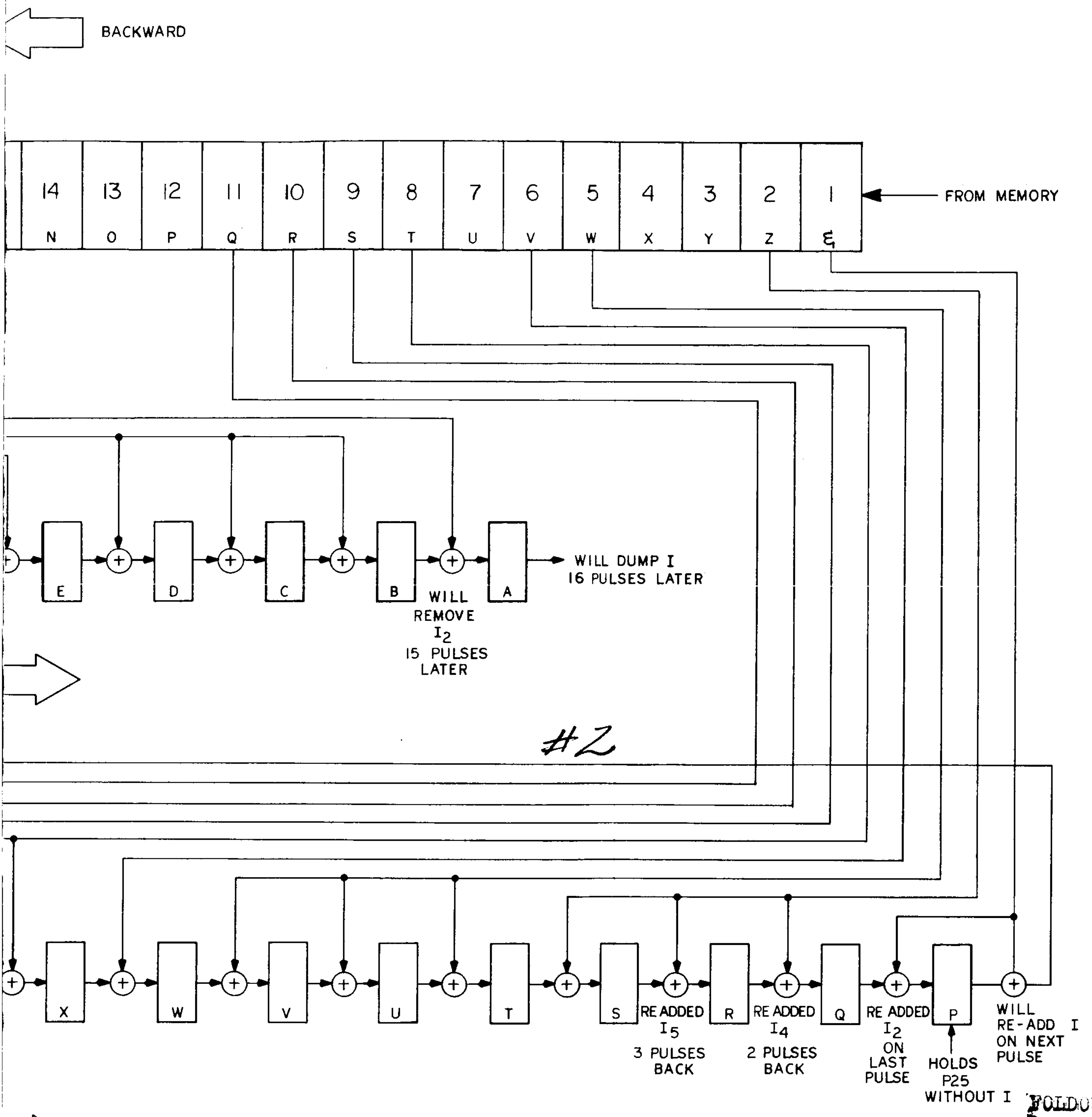
TO TABLE I_{26}
TO TABLE P_{26}
A
B
25 C
24 D
23 E
22 F
21 G
20 H
19 I
18 J
17 K
16 L
15 M
O
N
M
L
K
J
I
H
G
F
HOLDS P26 IN FULL
WILL REMOVE I_{25} 2 PULSES LATER
HOLDS P27 WITH-OUT I_{27}
WILL REMOVE I_{24} 3 PULSES LATER
HOLDS P28 WITH-OUT I_{28}, I_{27}
BACKWARD
AC
AB
AA
Z
Y
RE-ADDED I_{25} 16 PULSES BACK
RE-ADDED I_{24} 15 PULSES BACK
BACKWARD
$F_O(\bar{Q}_Y + Q\bar{Y}) + B_A(\bar{S}_Z + S\bar{Z})$
R
CLOCK

#/

TYPICAL STAGE

FOLDOUT FRAME 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



BACKWARD
14 N
13 O
12 P
11 Q
10 R
9 S
8 T
7 U
6 V
5 W
4 X
3 Y
2 Z
1 ε
FROM MEMORY
E
D
C
B
A
WILL DUMP I 16 PULSES LATER
WILL REMOVE I_2 15 PULSES LATER
#2
X
W
V
U
T
S
R
Q
P
RE ADDED I_5 3 PULSES BACK
RE ADDED I_4 2 PULSES BACK
RE ADDED I_2 ON LAST PULSE
HOLDS P25 WITHOUT I
WILL RE-ADD I ON NEXT PULSE

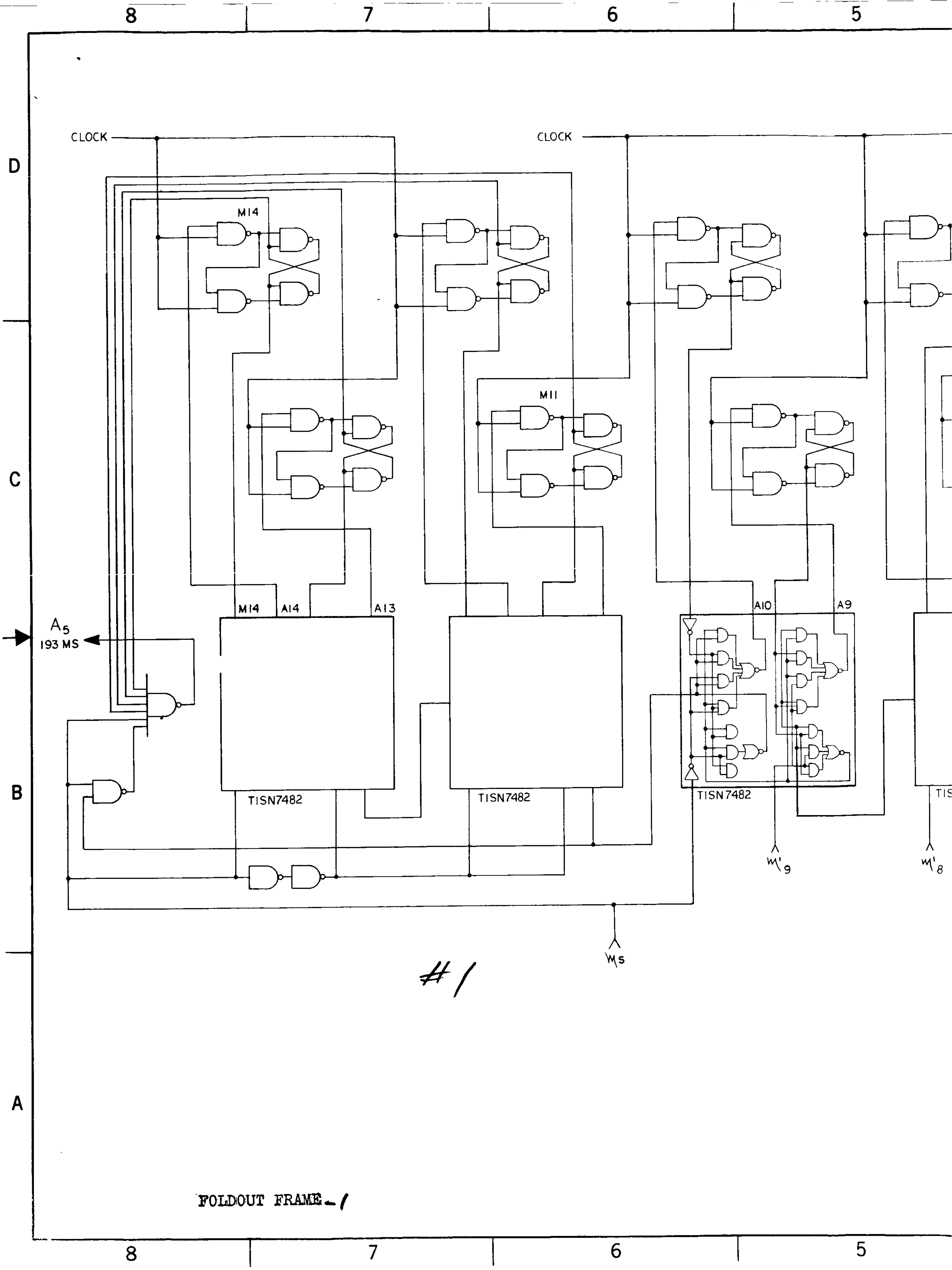
FOLDOUT [illegible] -2

					KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °-VENDOR ITEM-

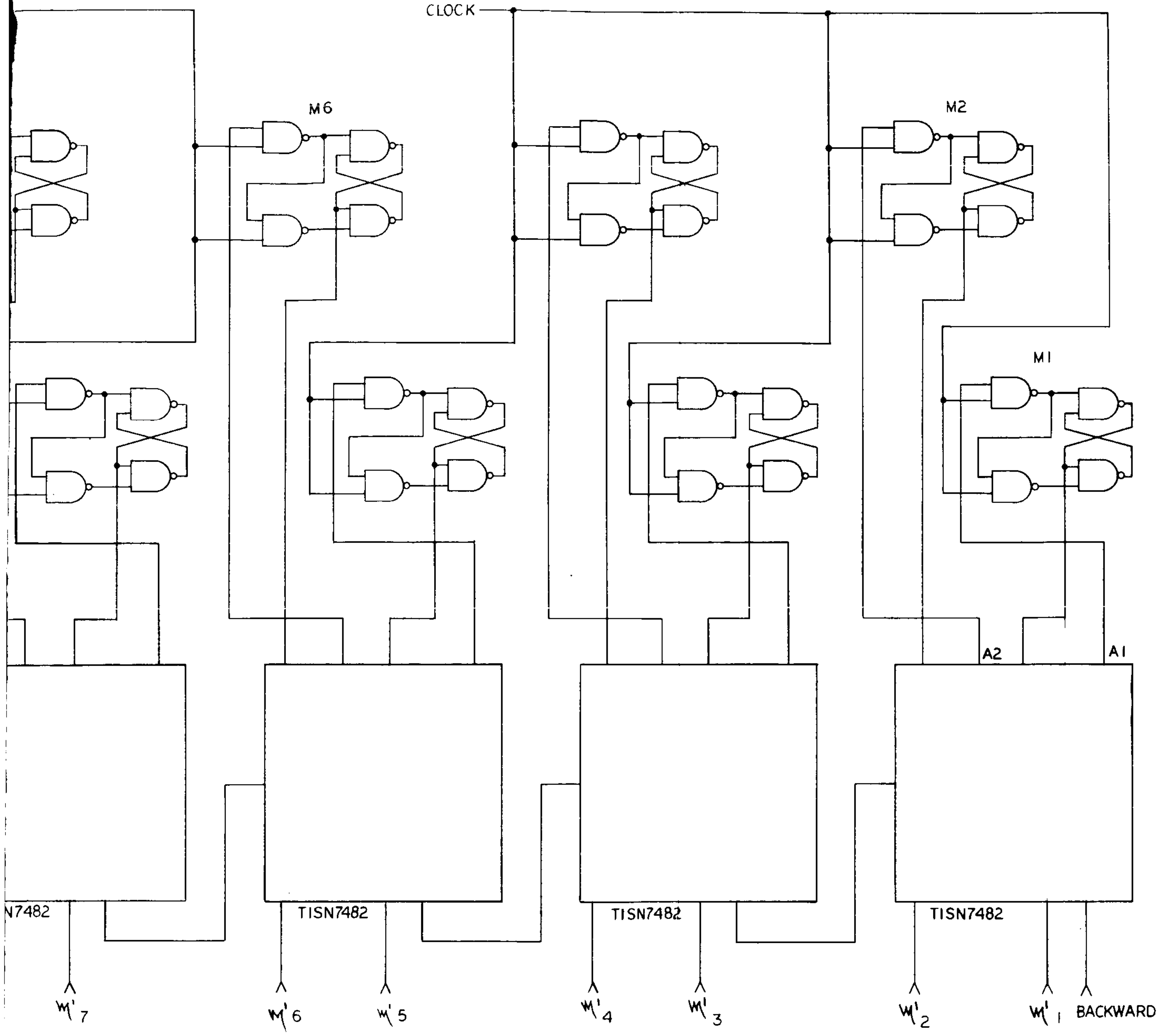
APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± SURFACES √ MICRO INCHES	DR A.W KELLAWAY DATE 5-1-67 CHK APPD [illegible] 5-2-67 APPD	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
FINISH	MATERIAL		SEQUENTIAL DECODER BACKWARD PARITY COMPUTATION
NEXT ASSEMBLY / MODEL			SIZE D / CODE IDENT 25420 / F028-7.00
			SCALE NONE / SHEET 1 OF 1



FOLDOUT FRAME-1

4	3	2	1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED



D

C

B

A

#2

FOLDOUT FRAME -2

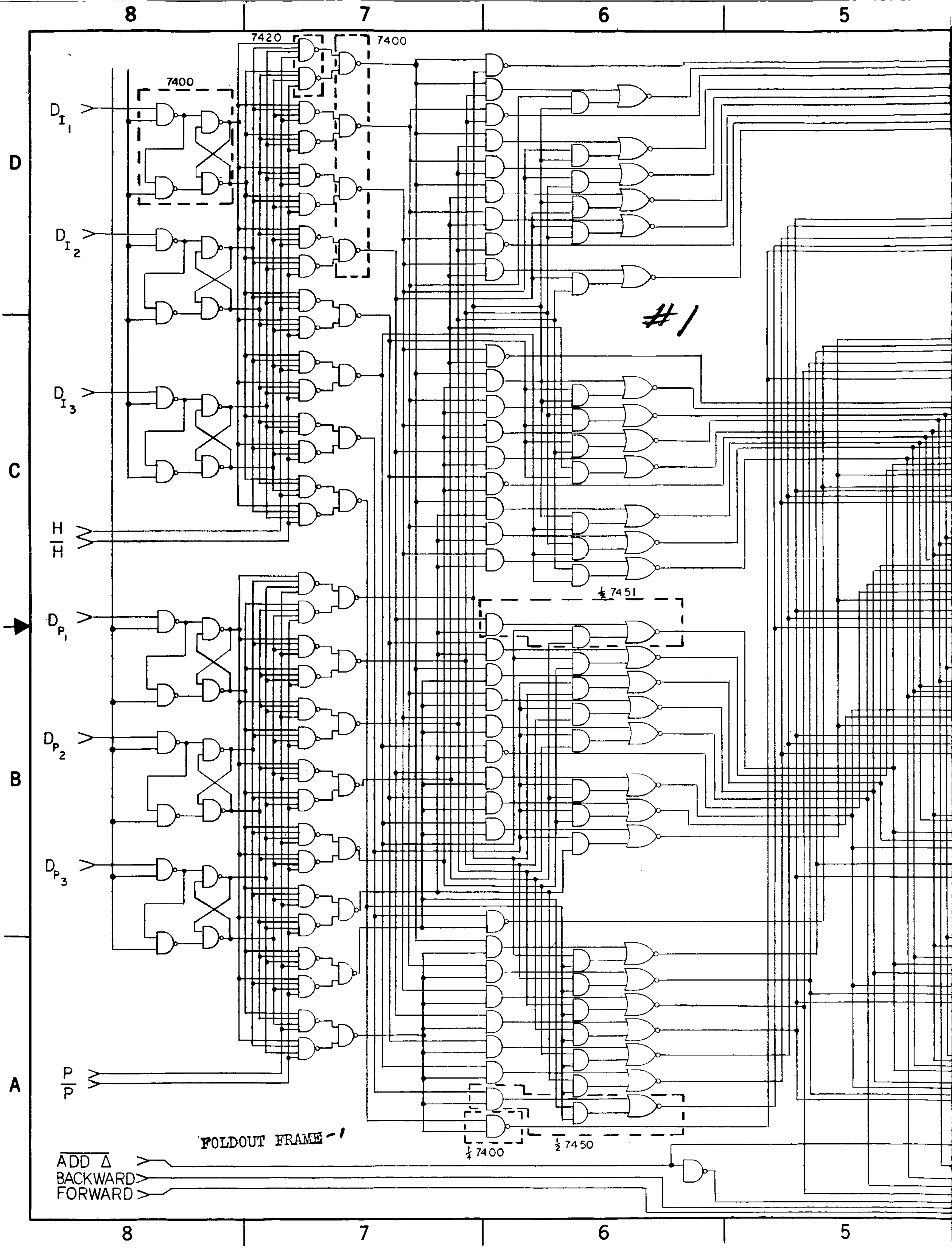
				KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

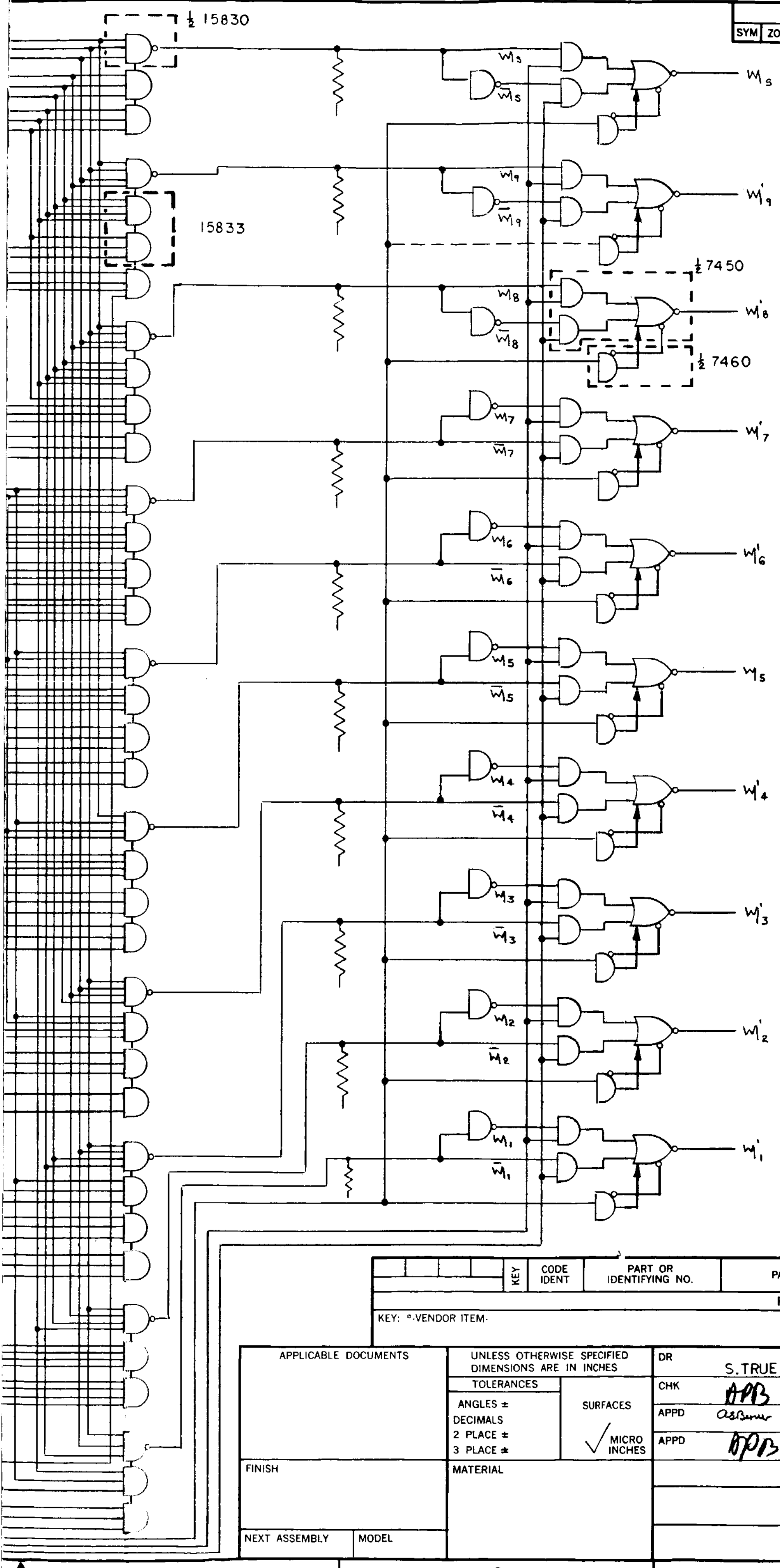
KEY: *VENDOR ITEM

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR A.W. KELLAWAY DATE 4-27-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± — SURFACES ✓ MICRO INCHES	CHK APB 4-30-67	SEQUENTIAL DECODER ADDER
		APPD [illegible] 4-30-67	
		APPD APB 4-30-67	
FINISH	MATERIAL		SIZE D — CODE IDENT 25420 — F028-08.00
NEXT ASSEMBLY — MODEL			SCALE NONE — SHEET 1 OF 1

4	3	2	1



8
7
6
5
D
C
B
A
7420
7400
7400
D_{I_1}
D_{I_2}
D_{I_3}
H
$\overline{H}$
D_{P_1}
D_{P_2}
D_{P_3}
P
$\overline{P}$
#1
½ 7451
¼ 7400
½ 7450
FOLDOUT FRAME-1
$\overline{ADD\ \Delta}$
BACKWARD
FORWARD



REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

#2

FOLDOUT FRAME - 2

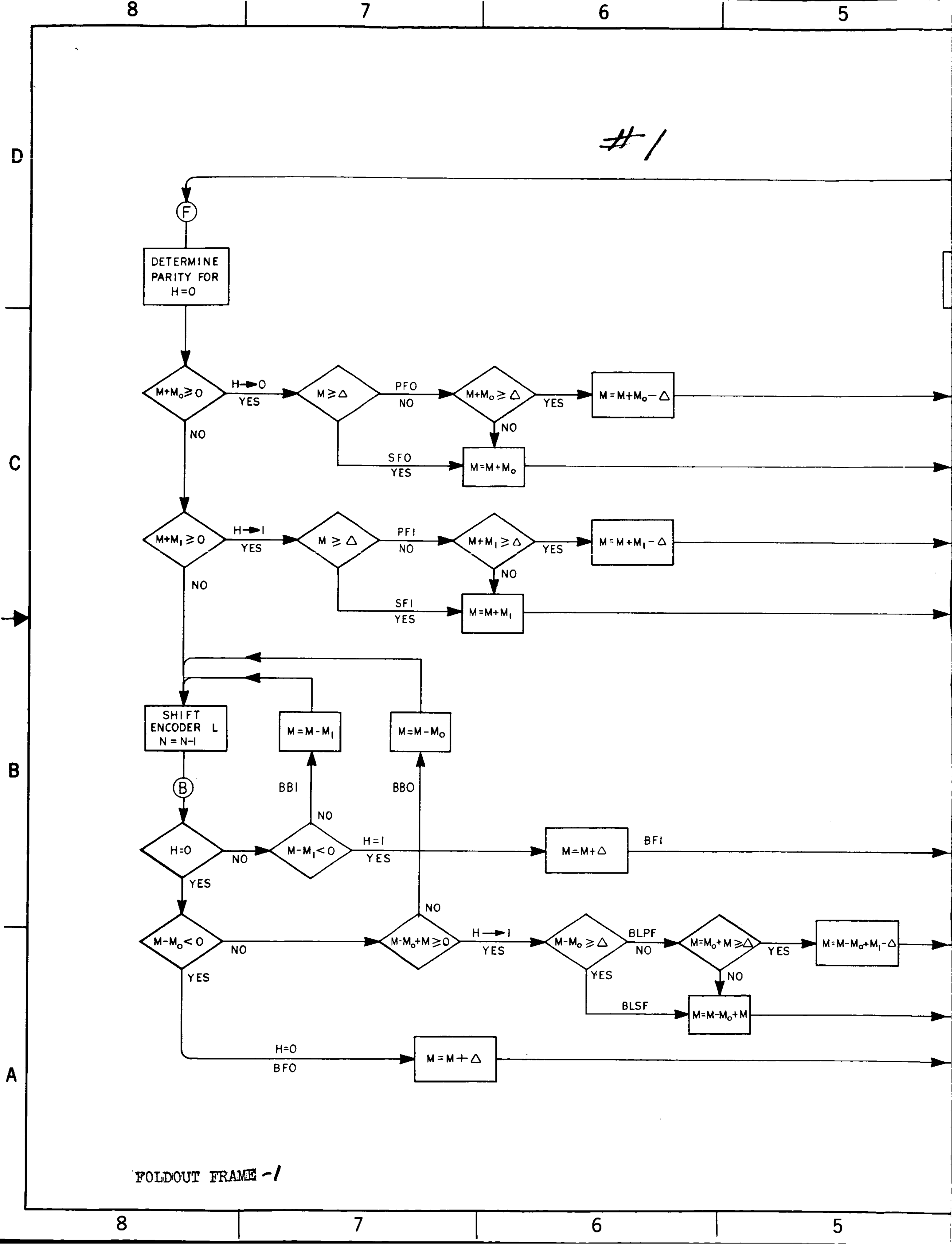
				KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °·VENDOR ITEM·

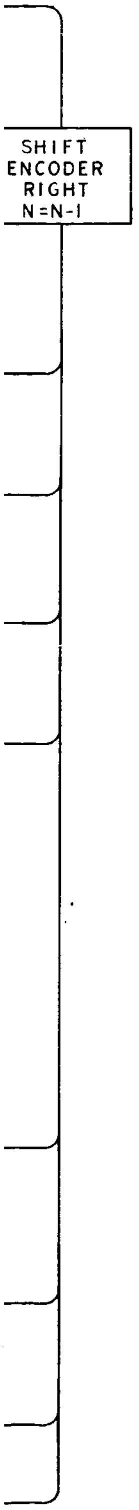
APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± SURFACES √ MICRO INCHES	DR S. TRUE DATE 4-28-67 CHK APB 9-30-67 APPD [illegible] 9-30-67 APPD APB 9-30-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS SEQUENTIAL DECODER METRIC TABLE
FINISH	MATERIAL		SIZE D · CODE IDENT 25420 · F028.09.00
NEXT ASSEMBLY · MODEL			SCALE — · SHEET 1 OF 1

#/



FOLDOUT FRAME -1

#2



REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

FOLDOUT FRAME -2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± SURFACES ✓ MICRO INCHES	DR AW. KELLAWAY DATE 4-26-67 CHK [illegible] 4-30-67 APPD [illegible] 4-30-67 APPD [illegible] 4-30-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS SEQUENTIAL DECODER ALGORITHM
FINISH	MATERIAL		SIZE D · CODE IDENT 25420 · F028-10.00
NEXT ASSEMBLY · MODEL			SCALE NONE · SHEET 1 OF 1

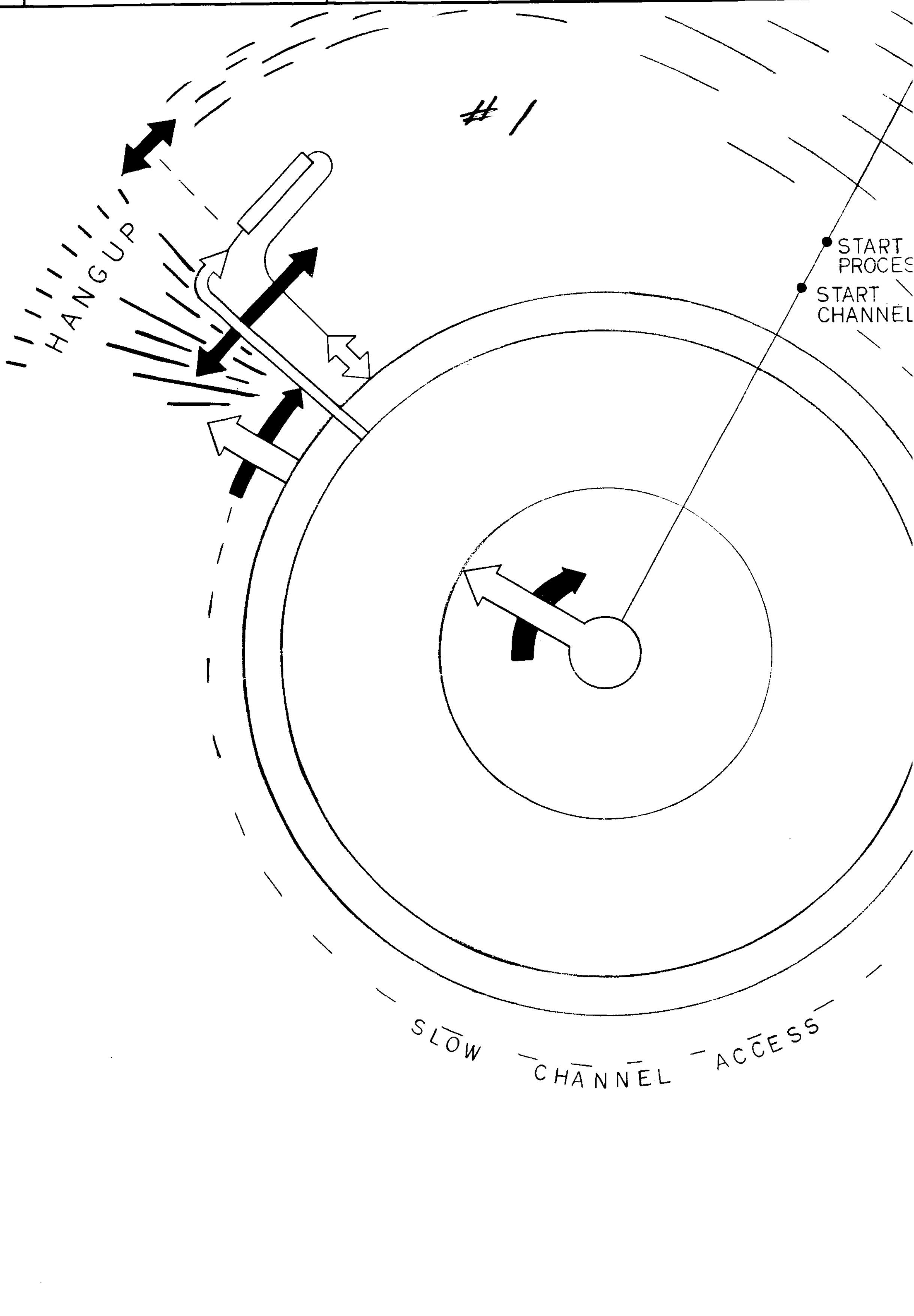
8 7 6 5

D

C

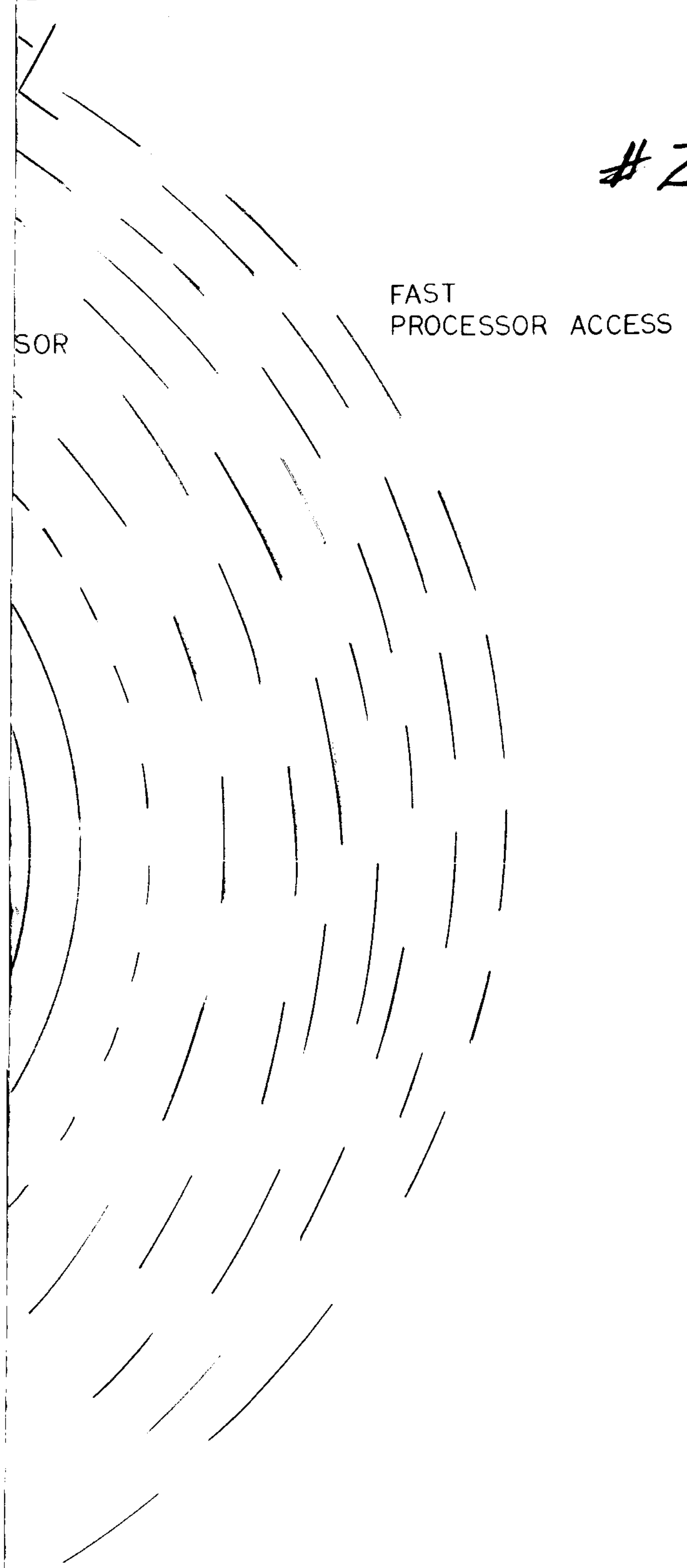
B

A



FOLDOUT FRAME -1

#2



FOLDOUT FRAME -2

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

					KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °·VENDOR ITEM·

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S. TRUE DATE 4-29-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± SURFACES √ MICRO INCHES	CHK	SEQUENTIAL DECODER HARE & TORTOISE RELATION
		APPD [illegible] 5-1-67	
		APPD	
FINISH	MATERIAL		SIZE D CODE IDENT 25420 F028-11.00
NEXT ASSEMBLY / MODEL			SCALE — SHEET 1 OF 1

8 7 6 5

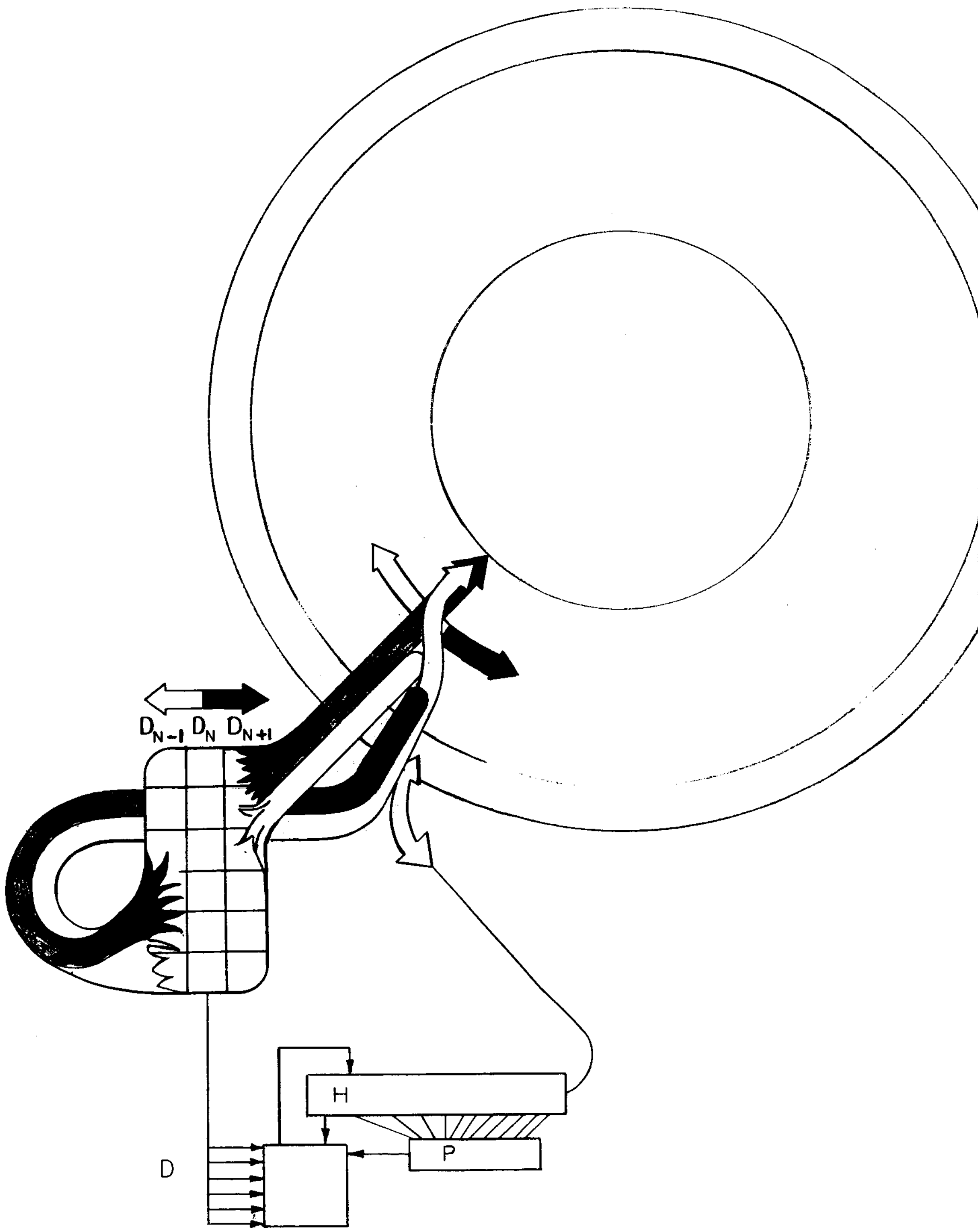
#1

D

C

B

A



FOLDOUT FRAME -1

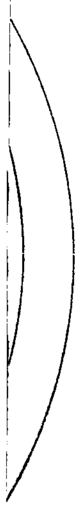
8 7 6 5

4	3	2	1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

A2

D



C

B

FOLDOUT FRAME-2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: °-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S TRUE DATE 4-29-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± / SURFACES √ MICRO INCHES	CHK	SEQUENTIAL DECODER FAST-ACCESS MODIFICATION
		APPD [illegible] 5-1-67	
		APPD	
FINISH	MATERIAL		SIZE D / CODE IDENT 25420 / F028-12.00
NEXT ASSEMBLY / MODEL			SCALE — / SHEET 1 OF 1

A

4	3	2	1

8 7 6 5

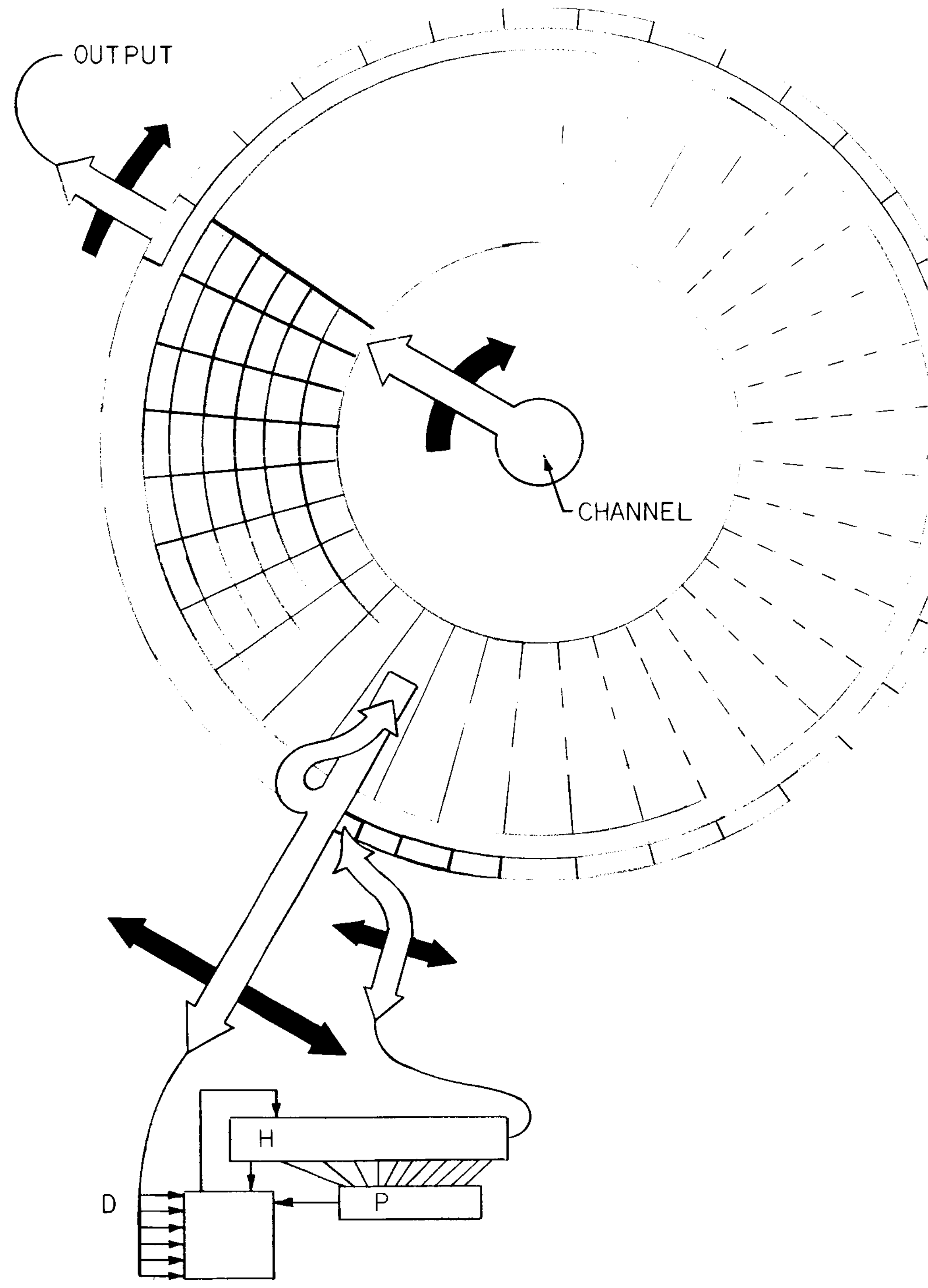
#/

D

C

B

A



FOLDOUT FRAME -/

8 7 6 5

4 3 2 1

REVISIONS				
SYM	ZONE	DESCRIPTION	DATE	APPROVED

#2

D

C

B

FOLDOUT FRAME - 2

	KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

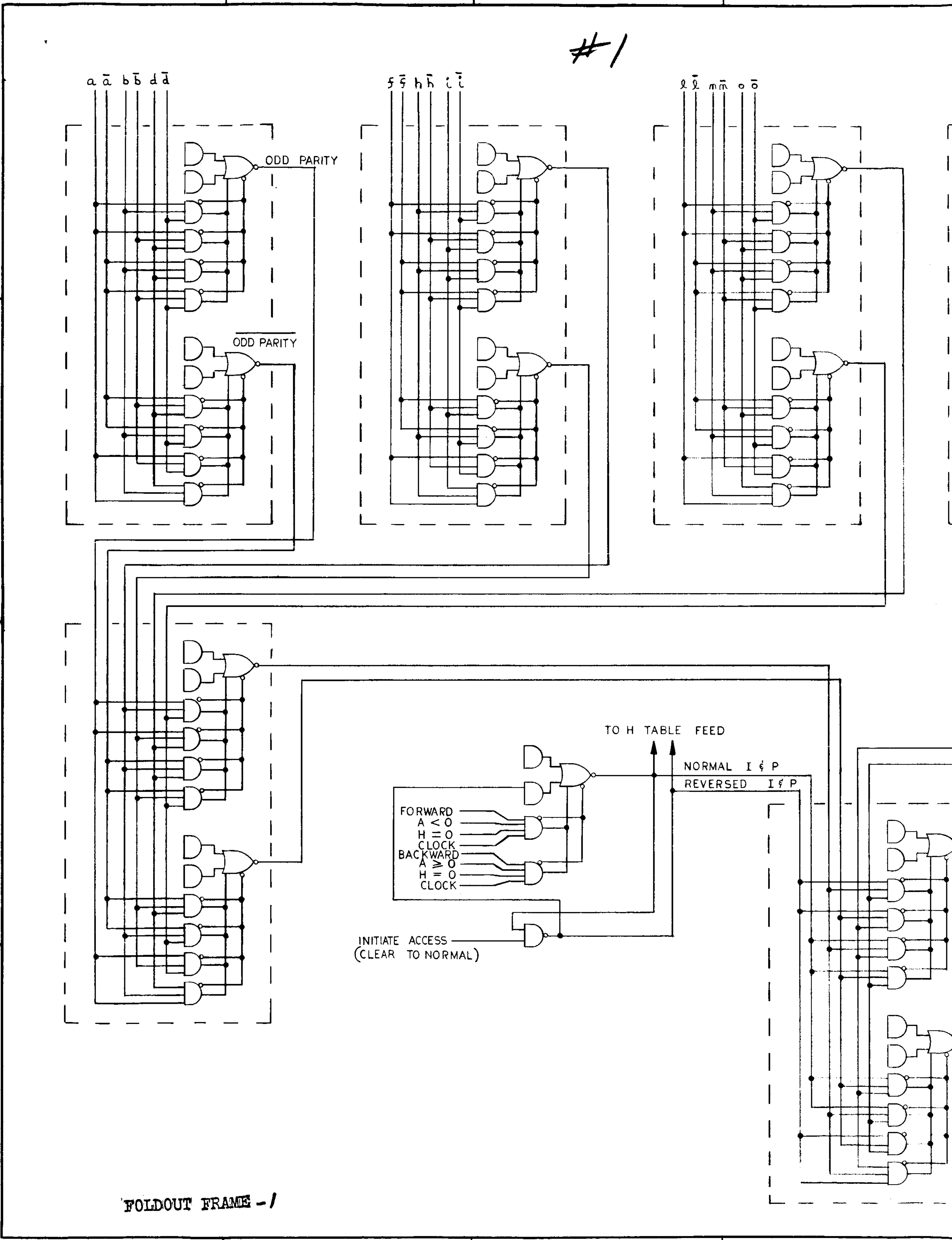
PARTS LIST

KEY: *-VENDOR ITEM-

A

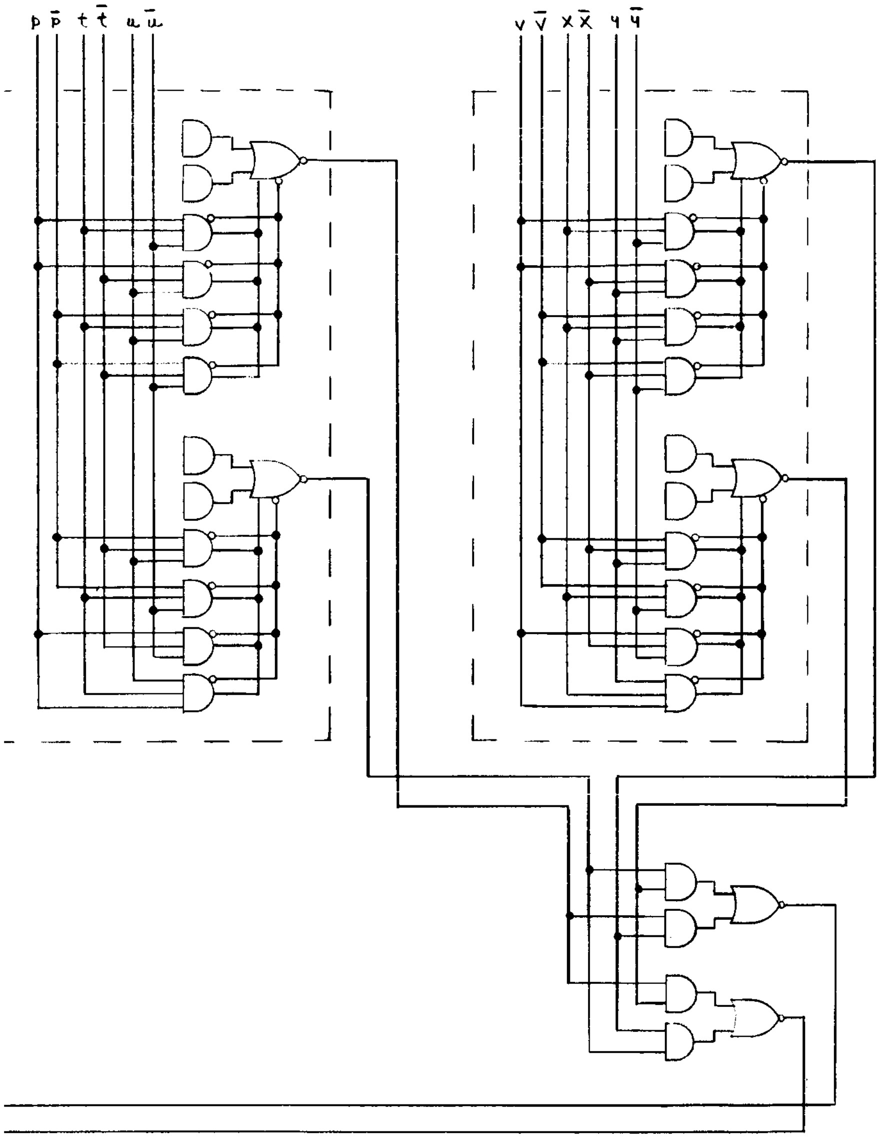
APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S. TRUE DATE 4-29-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± / SURFACES √ MICRO INCHES	CHK	
		APPD [illegible] 5-1-67	SEQUENTIAL DECODER MEMORY AS A 2-PORT RING
		APPD	
FINISH	MATERIAL		SIZE D / CODE IDENT 25420 / F028 - 13.00
NEXT ASSEMBLY / MODEL			SCALE — / SHEET 1 OF 1

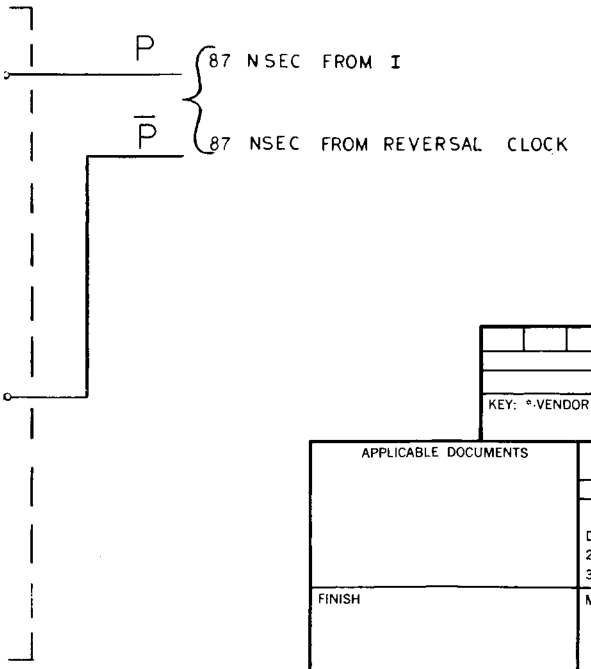
4 3 2 1



FOLDOUT FRAME - 1

#2





FOLDOUT FRAME -2

				KEY	CODE IDENT	PART OR IDENTIFYING NO.	PART NO.	NOMENCLATURE OR DESCRIPTION	NOTE	ITEM NO.

PARTS LIST

KEY: *-VENDOR ITEM-

APPLICABLE DOCUMENTS	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES	DR S. TRUE DATE 5-1-67	CODEX CORPORATION 222 ARSENAL STREET WATERTOWN, MASSACHUSETTS
	TOLERANCES: ANGLES ± DECIMALS 2 PLACE ± 3 PLACE ± — SURFACES √ MICRO INCHES	CHK	SEQUENTIAL DECODER PARITY GATING
		APPD [illegible] 5-1-67	
		APPD	
FINISH	MATERIAL		SIZE D — CODE IDENT 25420 — F028 -14.00
NEXT ASSEMBLY — MODEL			SCALE —— — SHEET 1 OF 1